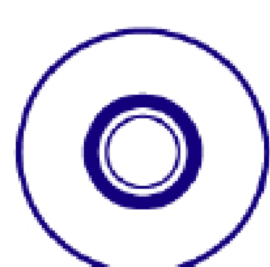
原创经典，程序员典藏

融汇9大热门开源技术，整合6大主流Java EE解决方案
精选19个典型模块和5个项目案例，实战Java Web整合开发

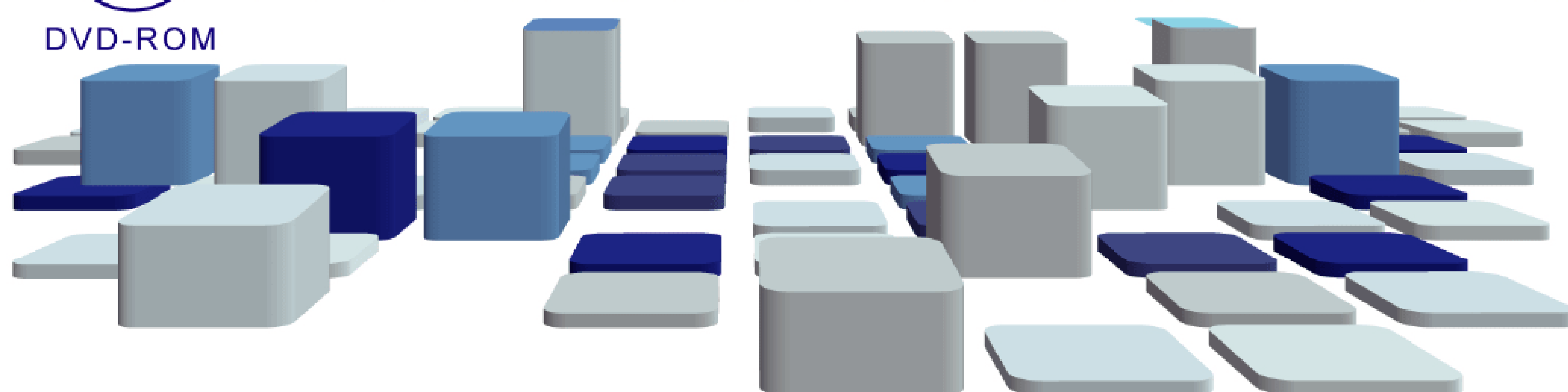
Java Web典型模块与项目实战大全

（49.5小时多媒体教学视频）

常建功　等编著



本书源代码、本书教学视频、赠送的Java学习视频……



清华大学出版社

Java Web 典型模块与项目实战大全

常建功　等编著

清华大学出版社
北　京

内 容 简 介

本书以实战开发为原则，以 Java EE 主流框架整合应用及项目开发为主线，通过 Java Web 开发中最常见的 19 个典型模块和 5 个完整的项目案例，详细介绍了 Struts 2.x、Spring、Guice、Hibernate、iBATIS、JPA、JSF 和 AJAX 等热门开源技术及 JSP+JavaBean+Servlet、Struts 2.x+Spring+Hibernate、Struts 2.x+Guice、Struts 2.x+Spring+JPA 和 Struts 2.x+Spring+iBATIS 等主流框架的整合使用。本书附带 1 张 DVD，内容为作者为本书录制的全程多媒体语音教学视频及本书所涉及的源代码。

本书分为 3 篇，共 27 章。涵盖的主要内容有：在线文本编辑器、验证模块、网络硬盘、网站统计模块、网络购物车、搜索引擎、在线网上支付、邮件发送系统、网络留言板、JQuery 框架经典应用、在线文件上传和下载、网上投票系统、商业银行网上账户管理系统、Hibernate 分页系统、生成报表、数据格式转换、用户维护功能、用户登录模块、在线音乐管理系统、数据汇聚系统、投票管理系统、权限管理系统、商业银行设备巡检系统等。

本书内容丰富，实例典型，实用性强，适合各个层次想要学习 Java Web 开发技术的人员阅读，尤其适合有一定 Java EE 基础而要进行 Web 应用开发的人员阅读。

图书在版编目（CIP）数据

Java Web 典型模块与项目实战大全 / 常建功等编著. —北京：清华大学出版社，2010.9
ISBN 978-7-302-22589-8

Ⅰ. ①J…　Ⅱ. ①常…　Ⅲ. ①JAVA 语言－程序设计　Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2010）第 078448 号

责任编辑： 夏兆彦
责任校对： 徐俊伟
责任印制：
出版发行： 清华大学出版社　　**地　址：** 北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编：** 100084
社 总 机： 010-62770175　　**邮　购：** 010-62786544
投稿与读者服务： 010-62795954，jsjjc@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销： 全国新华书店
开　本： 185×260　**印　张：** 59　**字　数：** 1473 千字
（附光盘 1 张）
版　次： 2010 年 9 月第 1 版　**印　次：** 2010 年 9 月第 1 次印刷
印　数： 1～ 000
定　价： 　元

产品编号：036230-01

前　言

为什么要写这本书

随着 Internet 的飞速发展，B/S 模式的软件开发越来越流行。由于 Java 语言在编写 B/S 模式的软件时具有得天独厚的优势，特别是 Internet 上多层结构应用系统的迅速流行，使得 Java EE 核心技术和各种解决方案得到了广泛的应用。程序员要想进入 Java EE 开发行业，除了需要有扎实的 Java 语言基础外，还要融会贯通各种开发框架，最好还要熟悉应用开发中有典型意义和实用价值的各类开发实例及案例。这样才能在就业严峻的市场环境中有较强的职场竞争力和职业前景。

目前图书市场上关于 Java Web 开发及框架整合的图书不少，但真正从实际应用出发，通过各种典型模块和项目案例来指导读者提高应用开发水平的图书却很少。本书便是以实战为主旨，通过 Java Web 开发中最常见的 19 个典型模块和 5 个完整的项目案例，让读者全面、深入、透彻地理解 Java Web 开发的各种热门技术及各种主流框架及其整合使用，从而提高实际开发水平和项目实战能力。

本书有何特色

1．附带多媒体语音教学视频，提高学习效率

为了便于读者理解本书内容，提高学习效率，作者专门为本书每一章的内容录制了大量的多媒体语音教学视频。这些视频和本书涉及的源代码一起收录于配书光盘中。

2．涵盖Java Web开发的各种热门技术及主流框架的整合使用

本书涵盖 Struts 2.x、Spring、Guice、Hibernate、iBATIS、JPA、JSF 和 AJAX 等热门开源技术，以及 JSP+JavaBean+Servlet、Struts 2.x+Spring+Hibernate、Struts 2.x+Guice、Struts 2.x+Spring+JPA 和 Struts 2.x+Spring+iBATIS 等主流框架的整合使用。

3．对Java Web开发的各种技术和框架作了原理上的分析

本书从一开始便对 Web 开发基础和 Java Web 开发的环境配置做了基本介绍，并对各种开发技术和主流框架及其整合进行了原理性的分析，便于读者理解书中后面的典型模块开发和项目案例。

4．模块驱动，应用性强

本书提供了 19 个 Web 开发的典型模块，这些模块都是 Java Web 开发中经常要用到的模块，具有超强的实用性，而且这些模块相互独立，应用开发人员可以随时查阅和参考。

5．项目案例典型，实战性强，有较高的应用价值

本书最后一篇提供了 5 个项目实战案例。这些案例来源于作者所开发的实际项目，具有很高的应用价值和参考性。而且这些案例分别使用不同的框架组合来实现，便于读者融会贯通地理解本书中所介绍的技术。这些案例稍加修改，便可用于实际项目开发中。

6．提供完善的技术支持和售后服务

本书提供了专门的技术支持邮箱：bookservice2008@163.com。读者在阅读本书过程中有任何疑问都可以通过该邮箱获得帮助。

本书内容及知识体系

第1篇　开发工具及框架概述（第1～3章）

本篇介绍了 Java EE 开发环境的配置和主流框架的基础知识。主要包括 Web 开发基础、配置 Java EE 开发环境、MyEclipse 开发工具对各种框架的支持、实现各种框架的集成等。

第2篇　典型模块开发（第4～22章）

本篇介绍了 Java Web 开发中最常用的 19 个典型模块的实现。主要包括在线文本编辑器、验证模块、网络硬盘、网站统计模块、网络购物车、搜索引擎、在线网上支付、邮件发送系统、网络留言板、JQuery 框架经典应用、在线文件上传和下载、网上投票系统、商业银行网上账户管理系统、Hibernate 分页系统、生成报表、数据格式转换、用户维护功能、用户登录模块等。

第3篇　项目案例实战（第23～27章）

本篇主要介绍了 5 个项目案例的开发过程。主要包括在线音乐系统、数据汇聚系统、投票管理系统、权限管理系统和商业银行设备巡检系统。在具体剖析这 5 个系统时涉及需求分析、数据库设计、持久层设计、业务层设计和表示层设计的详细过程。

配书光盘内容介绍

为了方便读者阅读本书，本书附带 1 张 DVD 光盘。内容如下：

- ❑ 本书所有实例的源代码；
- ❑ 本书每章内容的多媒体语音教学视频；
- ❑ 免费赠送的 Java Web 开发教学视频及相关电子书。

适合阅读本书的读者

- ❑ 需要全面学习 Java Web 开发技术的人员；
- ❑ 广大 Web 开发程序员；
- ❑ Java 程序员；
- ❑ Java EE 开发工程师；
- ❑ 希望提高项目开发水平的人员；
- ❑ 专业培训机构的学员；
- ❑ 软件开发项目经理；
- ❑ 需要一本案头必备查询手册的人员。

阅读本书的建议

- ❑ 没有 Java EE 框架基础的读者，建议从第 1 章顺次阅读并演练每一个实例；
- ❑ 有一定 Java EE 框架基础的读者，可以根据实际情况有重点地选择阅读各个模块和项目案例；
- ❑ 对于每一个模块和项目案例，先自己思考一下实现的思路，然后再阅读，学习效果会更好；
- ❑ 可以先对书中的模块和项目案例阅读一遍，然后结合光盘中提供的多媒体教学视频再理解一遍，这样理解起来就更加容易，也会更加深刻。

本书作者及编委会成员

本书由常建功主笔编写。其他参与编写和资料整理的人员有王征、王石、姜海英、邵毅、张路平、李臻、武勇、徐宁、刘玉珊、麻雪、吝晓宁、范永龙、赵盟、傅靖、李佳、刘丹、肖冰、王行恒、冯浩楠、纪超、段桂东、黄宝生、张珍珍、石淑珍、陈超、牛晓辉、刘聪、任潇、张双、于志华、李秀劲、李胜美、蔡文仙、杜阳阳、吴兴亮、陈水望、黄任桢、梅婷婷、皇波、白雪蛟。在此一并表示感谢！

本书编委会成员有欧振旭、陈杰、陈冠军、项宇峰、张帆、陈刚、程彩红、毛红娟、聂庆亮、王志娟、武文娟、颜盟盟、姚志娟、尹继平、张昆、张薛。

编著者

目　　录

第 1 篇　开发工具及框架概述

第 2 篇 典型模块开发

第 3 篇 项目案例实战



教学视频：26 分钟



教学视频：11 分钟

第 1 篇　开发工具及框架概述

第 1 章　开 发 前 奏

本章内容将讲解 Java Web 开发环境相关软件（JDK、MyEclipse 和 Tomcat）和数据库软件（MySQL 和 Oracle）的下载、安装和运行。本章的操作如果没有具体说明，均是在 Windows XP 操作系统下进行。在具体开发 Java Web 方面的项目时，除了需要掌握好基础知识（Servlet、Java Sever Page 和 JavaBean），还需要熟悉各种框架技术。

1.1　Java Web 应用概述

随着 Internet 的发展，绝大部分应用都由 C/S（客户端/服务器）架构转换成 B/S（浏览器/服务器）架构。为了让程序员更容易编写出 Java Web 应用程序，Java Web 应用经历了最初的 HTTP 协议到 Servlet、JSP 技术的应用，再到 J2EE 的过程。

1.1.1　Java Web 应用程序基础：HTTP 协议

所有的 Java Web 应用程序都是基于 HTTP 协议，那么究竟什么是 HTTP？HTTP 全称为 Hypertext Transfer Protocal，意思是超文本传输协议，主要用来定义客户端和服务器端的通信规范。

在 Java Web 应用中，客户端不仅可以从本地磁盘上打开网页文档，而且还可以通过 HTTP 网络协议从服务器上获取网页文档。客户端与服务器在具体交互时，首先两者需要建立 TCP 网络连接，接着客户端按照 HTTP 协议的规定向服务器发出请求信息，当服务器接收到客户端的请求后，再按照 HTTP 协议的要求将结果发送给客户端，具体过程如图 1.1 所示。



图 1.1　交互过程

当网页提交请求给服务器时，经常会用到 get()和 post()方法，它们是 HTTP 协议中两个最简单的方法。get()方法主要用于要求服务器获得一个资源或返回该资源；post()方法不仅可以请求某个资源，而且还可以向服务器发送一些表单数据。

为了让客户端能够定位到服务器中的资源，通过 URL 定义 Internet 上的 Web 服务器中的每一个网页文件。那么究竟什么是 URL？URL 全称为 Uniform Resource Locator，意思是统一资源定位符。URL 地址中包含网络协议、服务器主机名（IP 地址）、文件（其他

资源）路径和端口号。

🔔**说明：** http://127.0.0.1:8080/test/test.html 地址中，http 为网络协议，127.0.0.1 为服务器地址，8080 为端口号，/test/test.htm 为文件资源地址。

1.1.2　Java Web 容器（Servlet+JavaBean+JSP）

在 Internet 发展的初期，所有的 Java Web 应用包含的都是静态的 HTML 页面。所谓静态页面，是指把呈现给浏览者的信息固定写在 HTML 页面中，该页面不具备与用户交互的能力，即没有动态显示的功能。

随着时间的推移，动态页面逐渐取代了静态页面。对于动态页面，如果还使用简单的传统技术则显得有些无能为力。为了让 Java Web 应用中包含动态执行的页面，最早出现的是 CGI 技术方案，该技术方案使得服务器与客户端的交互不再需要使用静态的 HTML 页面。CGI 技术方案不仅可以把数据库中的信息呈现给浏览者，而且还可以将浏览者的请求保存到数据库中。虽然 CGI 技术方案开启了动态 Web 应用的时代，但是其却存在很多缺点，其中最大的缺点是不仅开发难度非常大，而且性能上也存在许多限制。

1997 年，在 Java 开发者的关注中，Servlet 技术终于诞生。该技术是 Sun 公司提供的一种动态页面的解决方案，实现 HTTP 协议在 Java 平台的一个扩展。

Servlet API 1.0 不仅能够开发 HTTP 协议方面的程序，而且还可以开发 Web Server、Mail Server、Ftp Server 和 Applicaton Server 等方面的服务，因此在编写服务器端的程序时都离不开 Servlet 语言。但是 Servlet 语言将程序的逻辑控制代码与输出网页文档内容混合在一起，使得控制网页文档内容的显示外观和整体布局很难。为了弥补 Servlet 语言的这些缺陷，Sun 公司又在该语言的基础上推出了 Java Sever Page（JSP）技术。

所谓 JSP 页面，就是在传统的 HTML 文件中加入 Java 程序片段和 JSP 标签。在该页面中可以通过 Java 程序片段操纵数据库、重定向网页等，实现建立动态网站所需要的功能。该页面的所有内容在服务器端执行，而传送给浏览者的仅为输出结果。使用 JSP 技术可以大大降低对客户端的要求。

为了提高代码的复用性、易维护性，Sun 公司又推出了 JavaBean 组件技术。从本质上讲 JavaBean 就是一个 Java 类，其有点类似于 Microsoft 的 COM 组件，主要用来描述 Java 的组件模型。

对于 Servlet、JSP 和 JavaBean 各种技术，它们本身并不会主动去处理各种请求，而是交给 Web 容器来管理。一个 Web 容器的结构如图 1.2 所示。

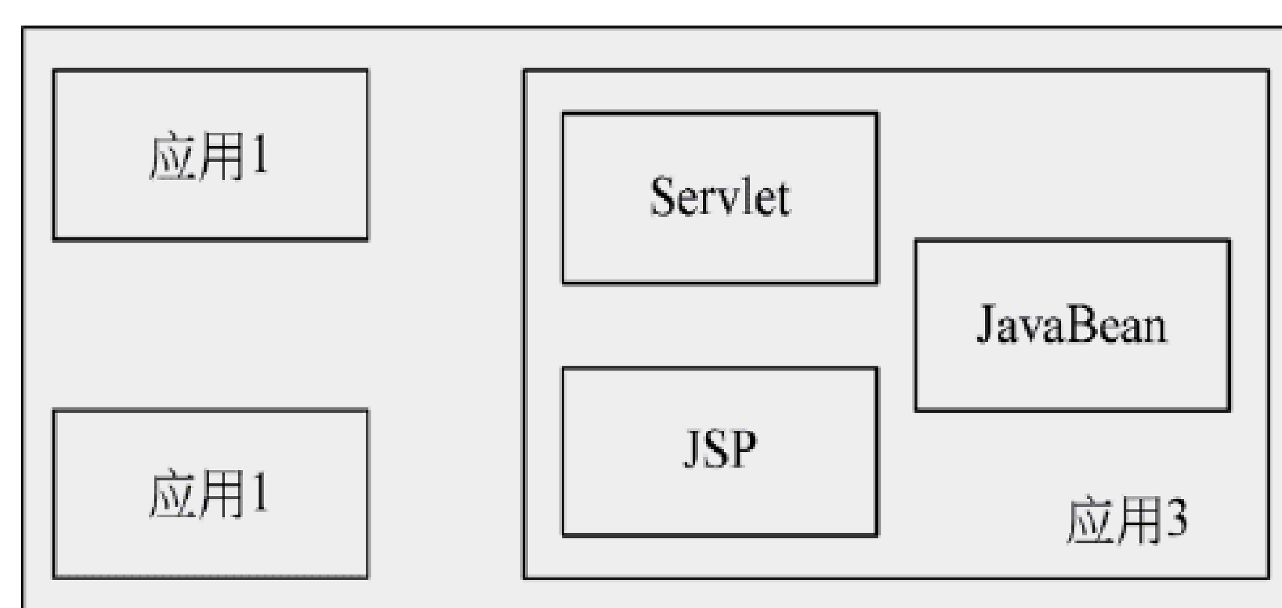


图 1.2　Web 容器结构

1.1.3　两种模式：Model 1 模式和 Model 2 模式

随着实际 Java Web 应用的使用越来越广泛，该种项目的规模不仅越来越大，而且其维护成本也越来越大。为了更好地使用动态 Java Web 编程技术，于是出现了所谓的 Model 1 和 Model 2 两个时代。

1. Model 1模式

由于 JSP 网页中经常使用业务逻辑（jsp:useBean）、流程管控（<%……%>）和 HTML 来开发系统，所以许多网站通常是通过一组 JSP 的结合开发出来，这种以 JSP 为核心的设计模式称为 Model 1。

根据 Model 1 的处理方式可以分为两类：一类是完全使用 JSP 开发；另一种则是使用 JSP+JavaBean 开发。

在图 1.3 的 JSP 模式中，其运行过程就是当浏览器发出一个请求到服务器端后，就由 JSP 来接收处理，最后服务器把 JSP 返回的结果回应给浏览器。虽然这种模式具有开发周期短、修改容易的优点，但是却降低了程序的可读性和复用性。

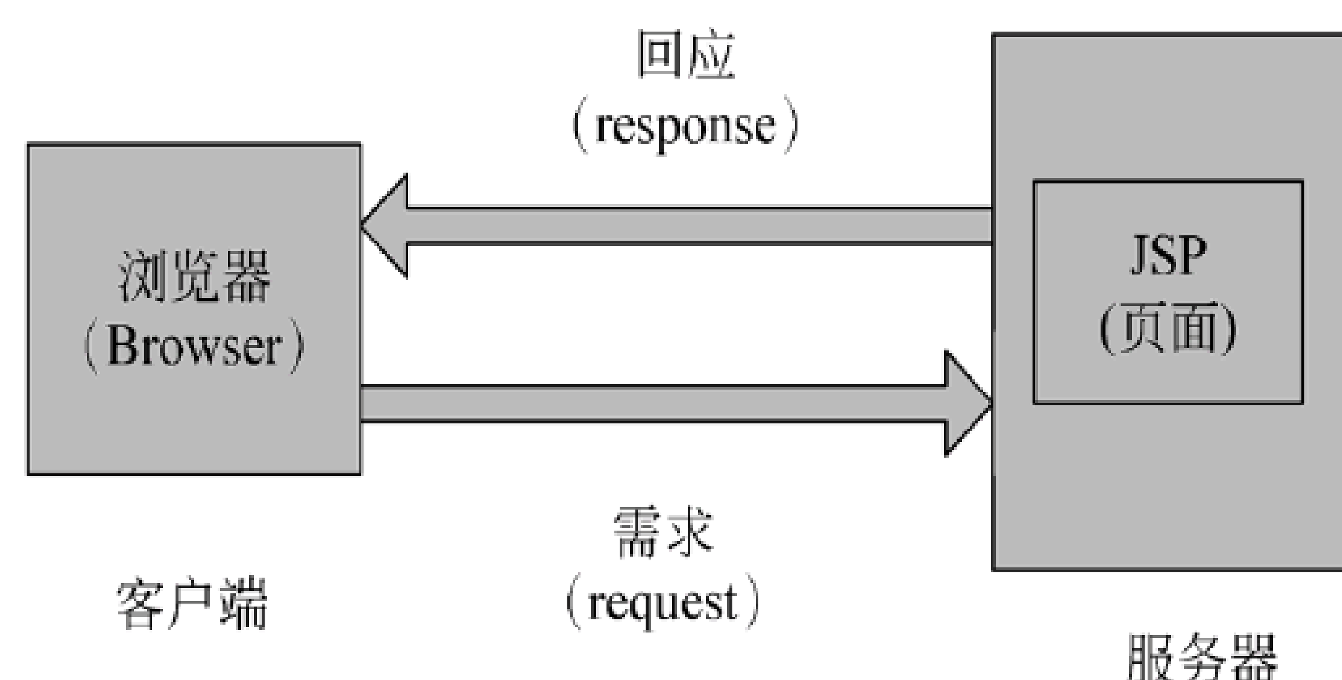


图 1.3　JSP 模式

在图 1.4 的 JSP+JavaBean 模式中，其运行过程就是当浏览器发出一个请求到服务器端后，就由 JSP 来调用相应的 JavaBean 负责处理，最后服务器把 JavaBean 返回的结果通过 JSP 页面回应给浏览器。虽然这种模式解决了 JSP 模式的程序可读性和复用性等缺点，但是缺乏流程控制。

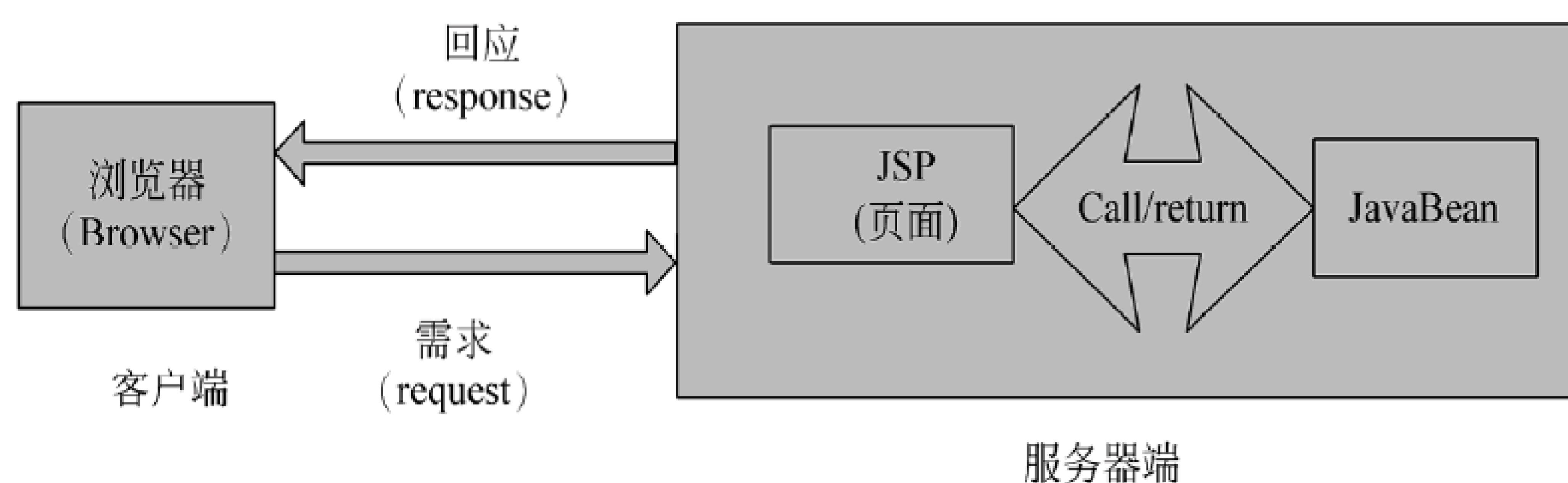


图 1.4　JSP+JavaBean 模式

注意： JSP+JavaBean 模式虽然也实现了页面的表现和页面商业逻辑的相分离，但是大量使用该模式，常常会导致页面被嵌入大量的脚本语言和 Java 代码。于是该模式不能够满足商业逻辑很复杂的大型项目。

2. Model 2模式

为了解决 JSP+JavaBean 模式缺乏流程控制的弊端，Model 2 整合了 Servlet 技术。在 Model 2 模式中由 Servlet 处理请求和控制业务流程，JSP 只负责输出回应给浏览器，而 JavaBean 负责具体的业务数据和业务逻辑，该模式如图 1.5 所示。

其实 Model 2 模式采用 MVC 架构作为开发模式，所以使得开发流程更为明确和维护更容易，但是对于开发者来说学习时间比较长，开发时间同样也比较长。

Model 2 与 Model 1 的本质区别是将处理请求的功能与产生显示内容的功能分配给两个独立的模块来完成。在 Model 2 中，由于 Servlet 不需要负责显示内容而 JSP 页面不需要实现任何业务流程和业务逻辑，因此不懂 Java 语言的普通 HTML 设计人员完全可以编写和维护 JSP 页面。这样就可以把程序开发者与网页制作人员有效地进行分离，让程序开发者专注于 Java 程序代码的编写，而 HTML 设计人员专注于页面的表现。

通过上面几节的讲解，可以发现 Java Web 编程技术经历了如图 1.6 所示的发展路线。

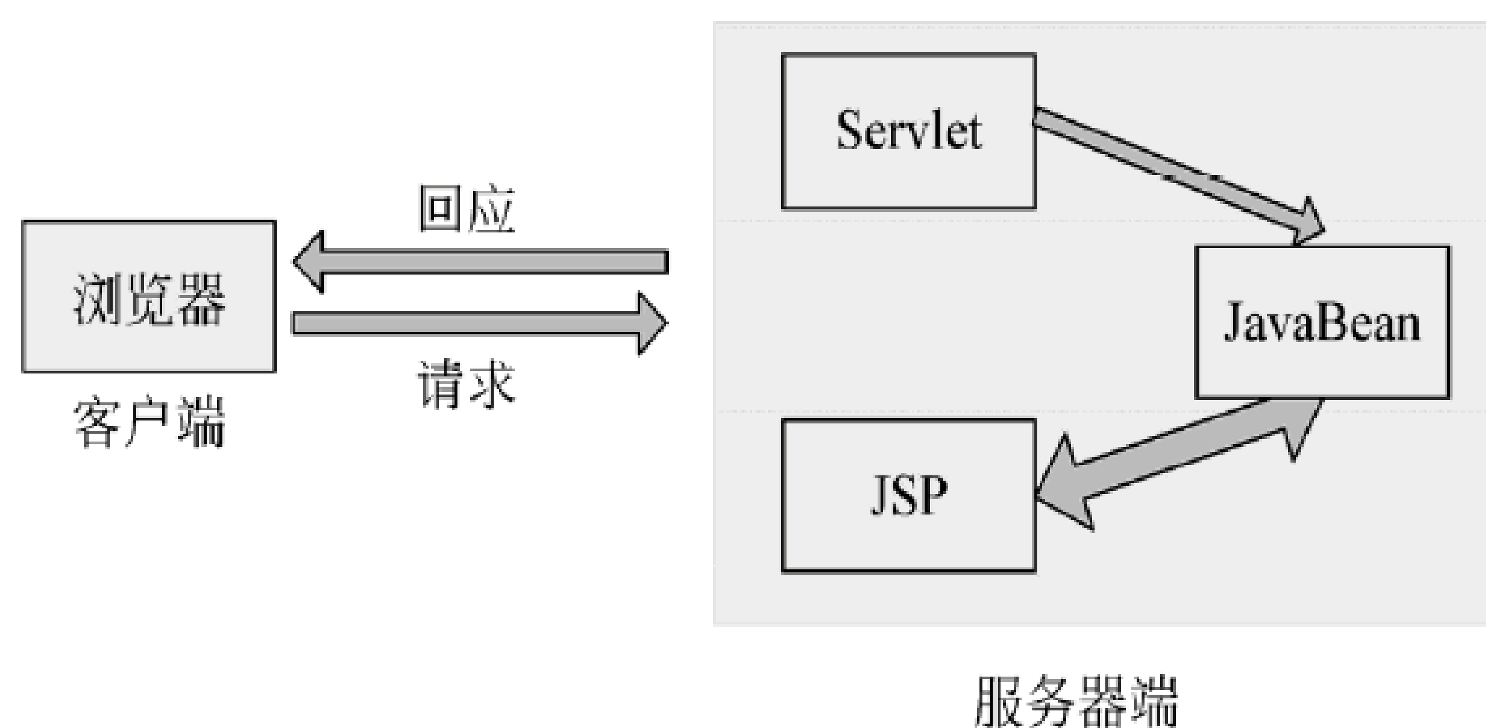


图 1.5　Servlet+JSP+JavaBean 模式

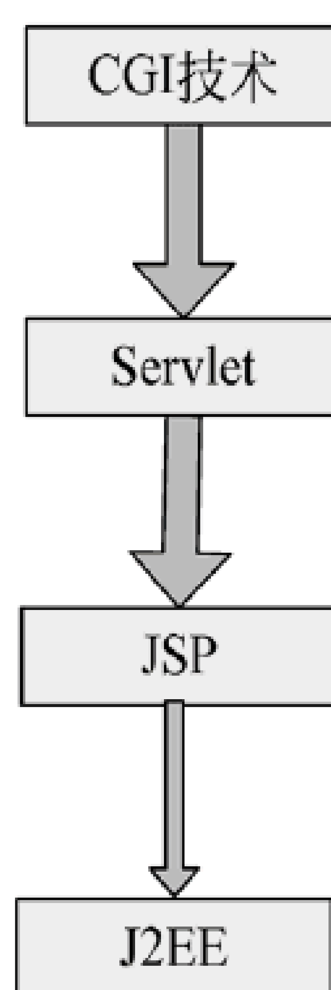


图 1.6　发展路线

1.1.4　MVC 设计思想

随着 J2EE 应用在 Internet 上的成熟，诞生了许多优秀的设计思想，MVC 就是其中的一个。由于 MVC 在理解和分析应用模型时提供了最基本的分析方法，在构造产品时提供了清晰的设计框架，所以 MVC 逐步成为 J2EE 平台上的主流设计思想。MVC 设计思想被广大开发人员认可并广泛应用至今，不停地被完善和发展，其中经历了几个不同的发展阶段。

1. 原始模式

最原始的 MVC 模式，如图 1.7 所示。在该模式中，浏览者会直接与视图进行会话。

该模式是具体流程，首先控制器获得客户端发出的视图信息，接着通过该信息实现对模型的相关操作，最后模型发生改动后以视图的形式将结果返回给客户端。

2．控制器模式

控制器的 MVC 模式，如图 1.8 所示。在该模式中，浏览者的请求将不会直接与视图进行会话，而是由一个分发器来接收。该模式具体流程是，首先分发器获得客户端发出的请求信息后，会将该请求转发给对应的控制器来处理，接着控制器会调用和修改模型，并返回一个对应某个视图的资源结果，最后将视图结果返回给客户端。

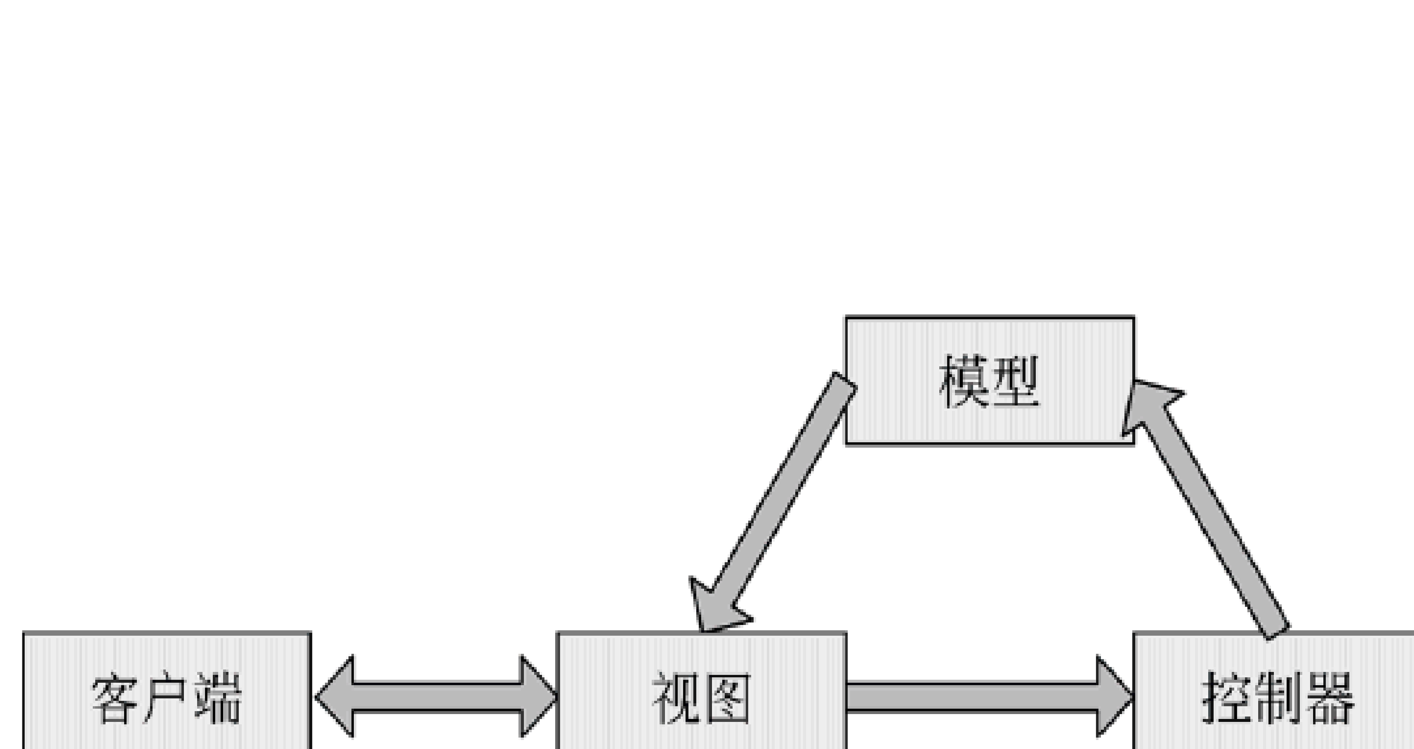


图 1.7　原始模式

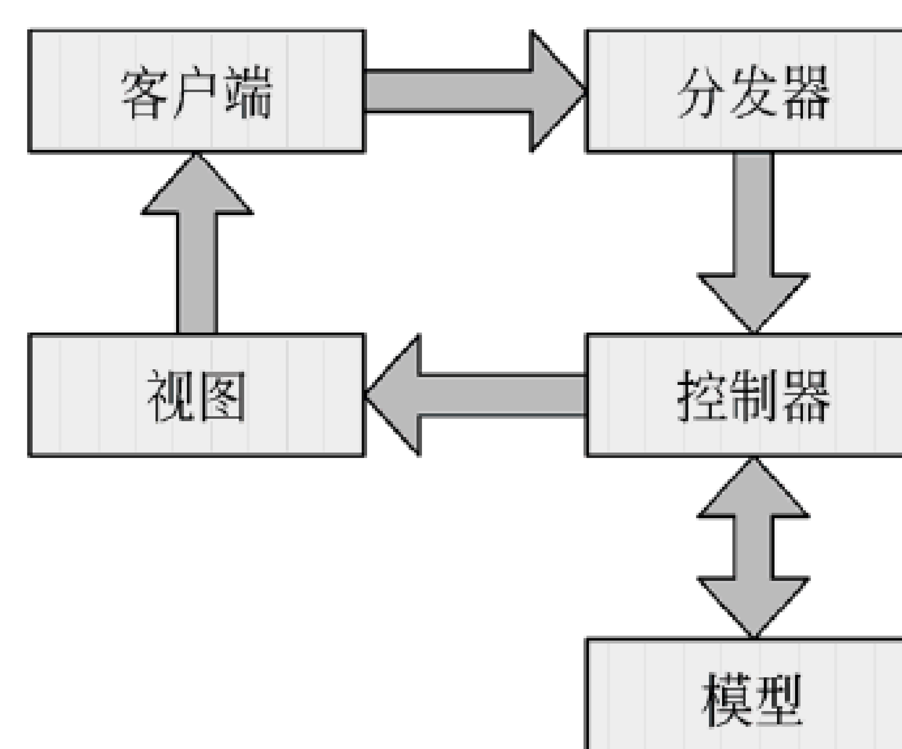


图 1.8　控制器模式

3．页面控制器模式

控制器的 MVC 模式如图 1.9 所示。在该模式中，会在生成视图之前调用相应的控制器，而不是通过分发器寻找控制器。

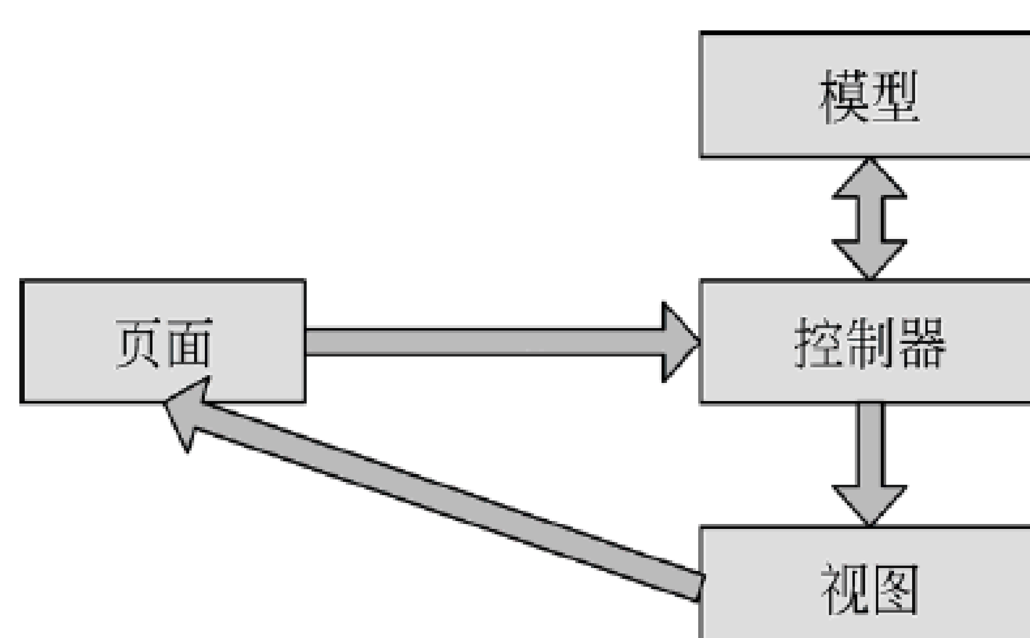


图 1.9　页面控制器模式

1.2　配置开发环境

在具体讲解相关 Java Web 应用系统技术之前，首先需要配置 Java Web 应用系统的开发环境。在具体配置开发环境时，涉及各种软件：开发工具包 JDK、服务器 Tomcat、IDE 工具 MyEclipse、数据库软件 MySQL 和 Oracle。

1.2.1　下载工具包 JDK

JDK（Java(TM)SE Development Kit）全称是 Java 标准版开发工具包，是 Java 开发和运行的基本平台。Java 语言程序代码的运行离不开该 JDK，使用其可以编译 Java 源代码为类文件。目前最稳定的版本为 JDK 6.0，注意下载时不要选择 Java 运行时环境（Java Runtime Environment，JRE），因为该种版本不包含 Java 编译器和 JDK 类源码。具体的下载步骤如下。

（1）首先访问下载 JDK 的官方网站（http://java.sun.com/javase/downloads/index.jsp），如图 1.10 所示。

（2）打开下载页后，单击相应版本后的 Download 按钮，如图 1.11 所示。这时就会弹出浏览器安全链接警告，在该对话框中单击“确定”按钮就会进入下载页面，如图 1.12 所示。



图 1.10　JDK 下载首页



图 1.11　选择 JDK 版本



图 1.12　JDK 下载页面

（3）在下载页面中首先选择要安装的平台，JDK 能支持多个主流操作系统，包括 Windows、Linux、Solaris 操作系统。而各种操作系统又根据 CPU 分成两种：X86 系列针对家用电脑的 31 位 CPU；X64 系列针对 64 位 CPU。一般只关注 Windows X86 版本即可。所以，在 Plattom 下拉列表框中选择 Windows 选项。在 Language 下拉列表框中选择 Multi-Language 选项，表示该软件包支持多国语言。完成上面步骤后，还必须选择下面的复选框表示接受下载协议。单击 Continue 按钮后，就会转到 JDK 安装文件的页面，如图 1.13 所示。在该页面中单击 jdk-6u16-windows-i586.exe 链接就会自动下载该软件。

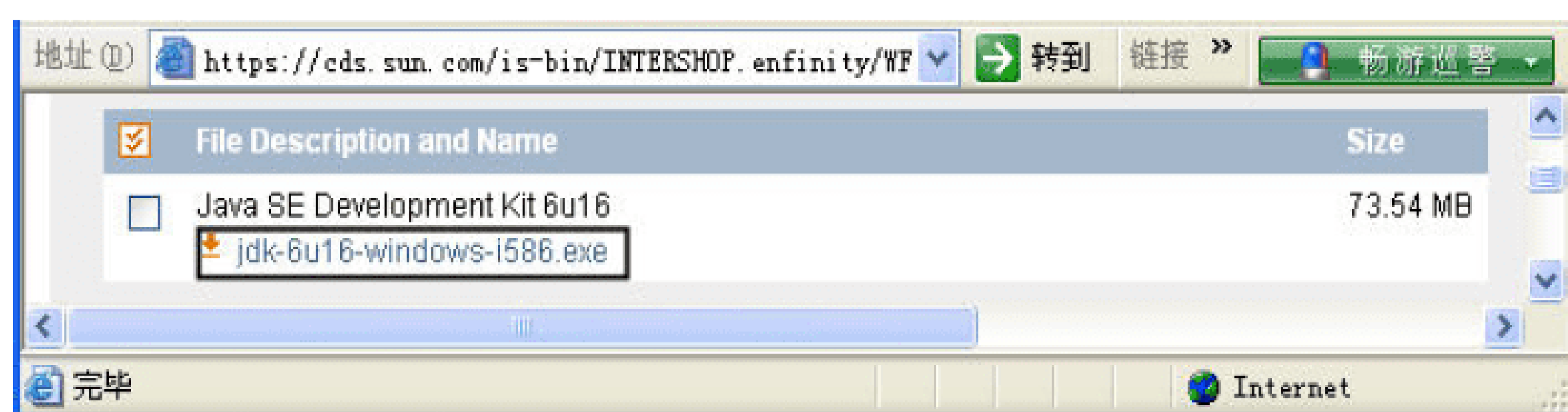


图 1.13　下载 JDK 安装文件

1.2.2　安装工具包 JDK

在 1.2.1 节介绍了如何下载 JDK 安装程序，下载完 JDK 安装程序后就可以开始安装 JDK 了。具体的安装步骤如下。

（1）双击 JDK 安装程序（jdk-6u10-windows-i586.exe），接着就会通过 Windows Installer 开始安装过程，如图 1.14 所示。

（2）先仔细阅读许可证协议，然后单击“接受”按钮。在弹出的自定义安装对话框中（如图 1.15 所示）可以选择安装内容和安装路径。

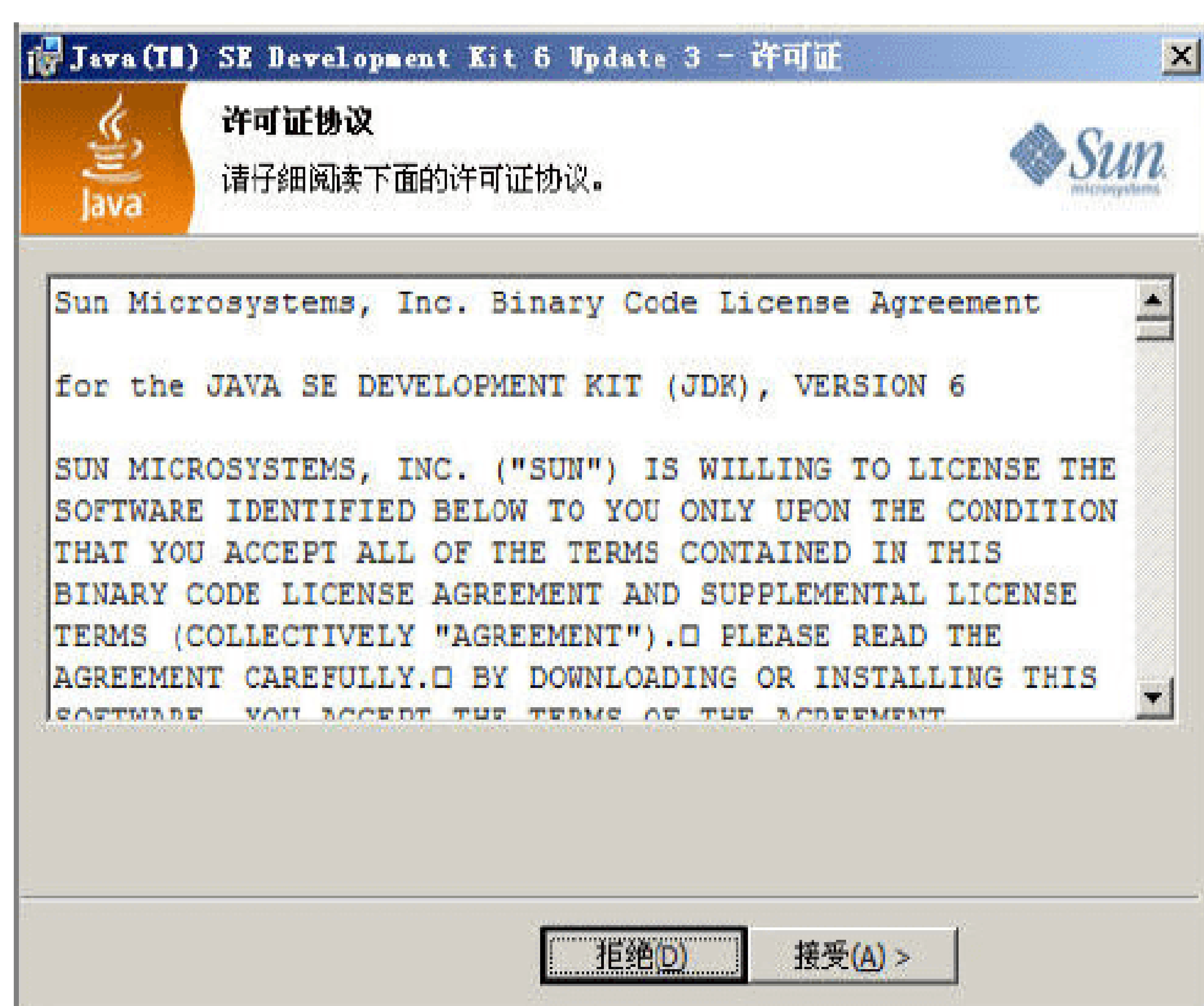




图 1.14　许可协议对话框　　　　图 1.15　自定义安装对话框

默认的安装内容如下。

- 开发工具：所谓的 JDK，是必须要安装的部分；
- 演示程序及样例：包含了代码的小程序和应用程序的演示和样例，建议初学者安装；
- 源代码：构成 Java 公共 API 类的源代码；
- 公共 JRE：独立的 JRE；
- Java DB：支持开源的 Java 技术数据库。

如果不想安装最后 3 项内容，可以单击选项前的下三角形按钮，在弹出的下拉列表框中选择“现在不安装此功能”选项，如图 1.16 所示。

一般推荐路径是“C:\jdk1.6.0_10\”，所以需要更改默认安装路径。单击“更改”按钮（如图 1.17 所示），然后在弹出的“更改当前目标文件夹”对话框中（如图 1.18 所示）选择相对应的路径。

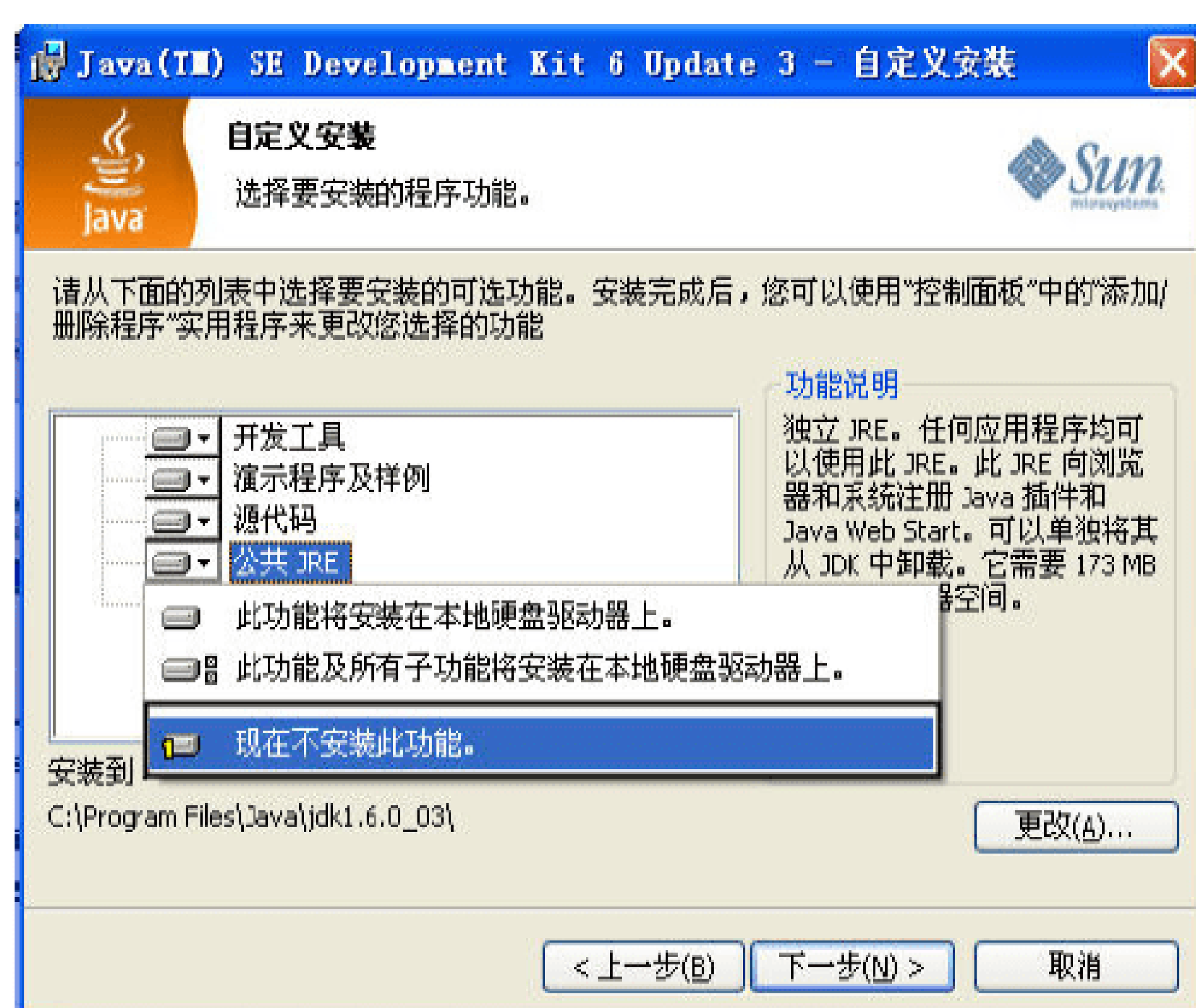


图 1.16 选择安装的程序内容



图 1.17 更改安装路径

注意：输入的路径中不推荐有空格和中文，之所以这样做是因为路径中有这些内容时会出现不必要的问题，导致某些 Java 程序运行失败。

（3）确认无误后，在“自定义安装”对话框中单击“下一步”按钮，开始执行安装程序。如果安装成功，会弹出如图 1.19 所示的对话框，然后单击“完成”按钮结束该 JDK 的安装。

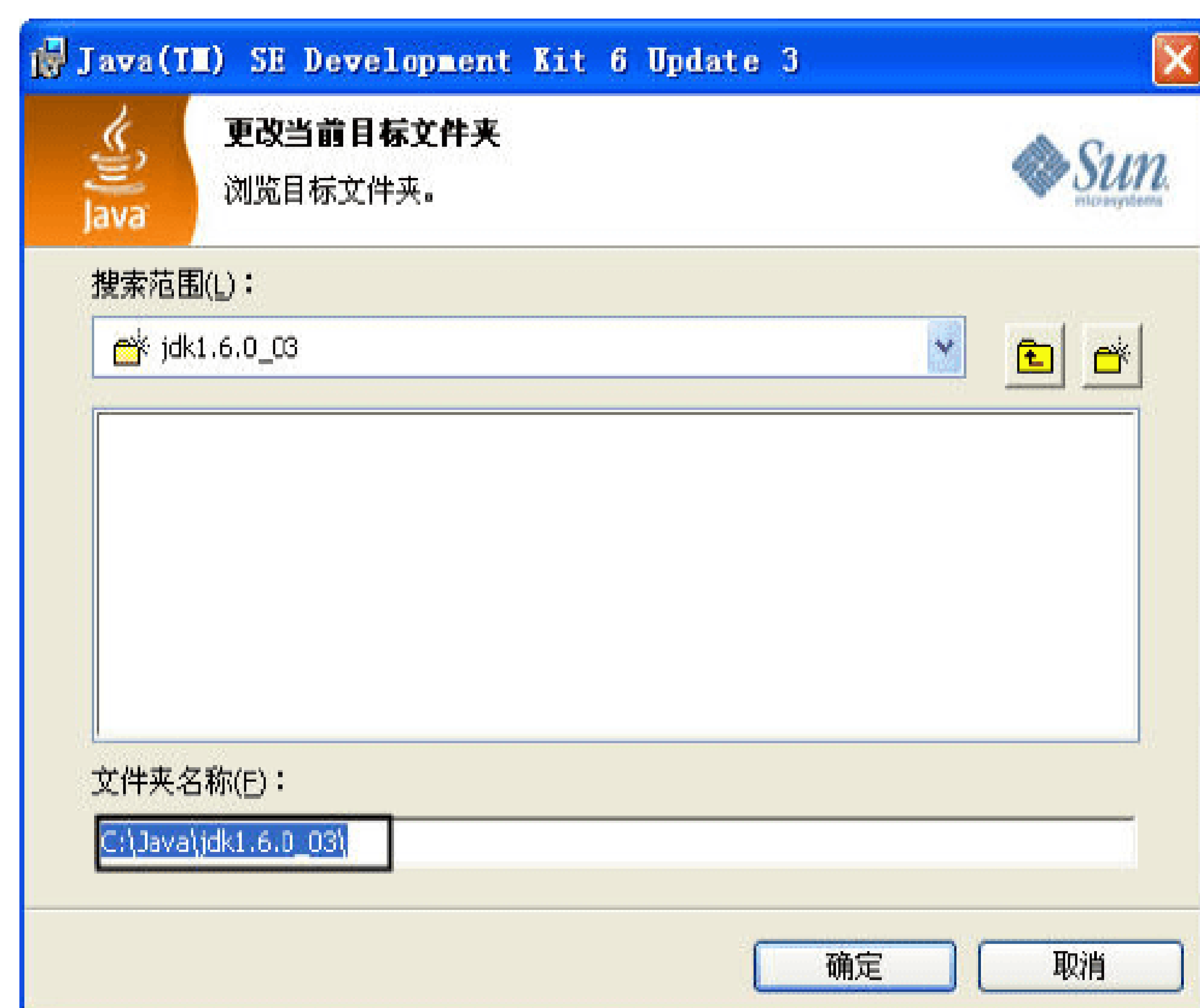


图 1.18 选择安装路径



图 1.19 完成 JDK 安装

1.2.3 下载服务器 Tomcat

Tomcat 是由 JavaSoft 和 Apache 共同合作出来的产品，是一款很不错的免费开源的 JSP 服务器，它被 Sun 公司推荐为运行 Servlet 和 JSP 的容器。同时要注意 Tomcat 还具有 Web 服务器的基本功能，能够提供数据库连接池、SSL、Proxy 等许多通用组件。Tomcat 目前最新的版本为 Tomcat 6.0.18，可以通过下面的步骤来实现该平台的下载。

（1）首先访问下载 Tomcat 的官方网站（http://tomcat.apache.org/），如图 1.20 所示。

（2）打开 Tomcat 首页后，单击页面中 Download 目录栏下的 Tomcat 6.x 链接，就会进

入下载页面，如图 1.21 所示。

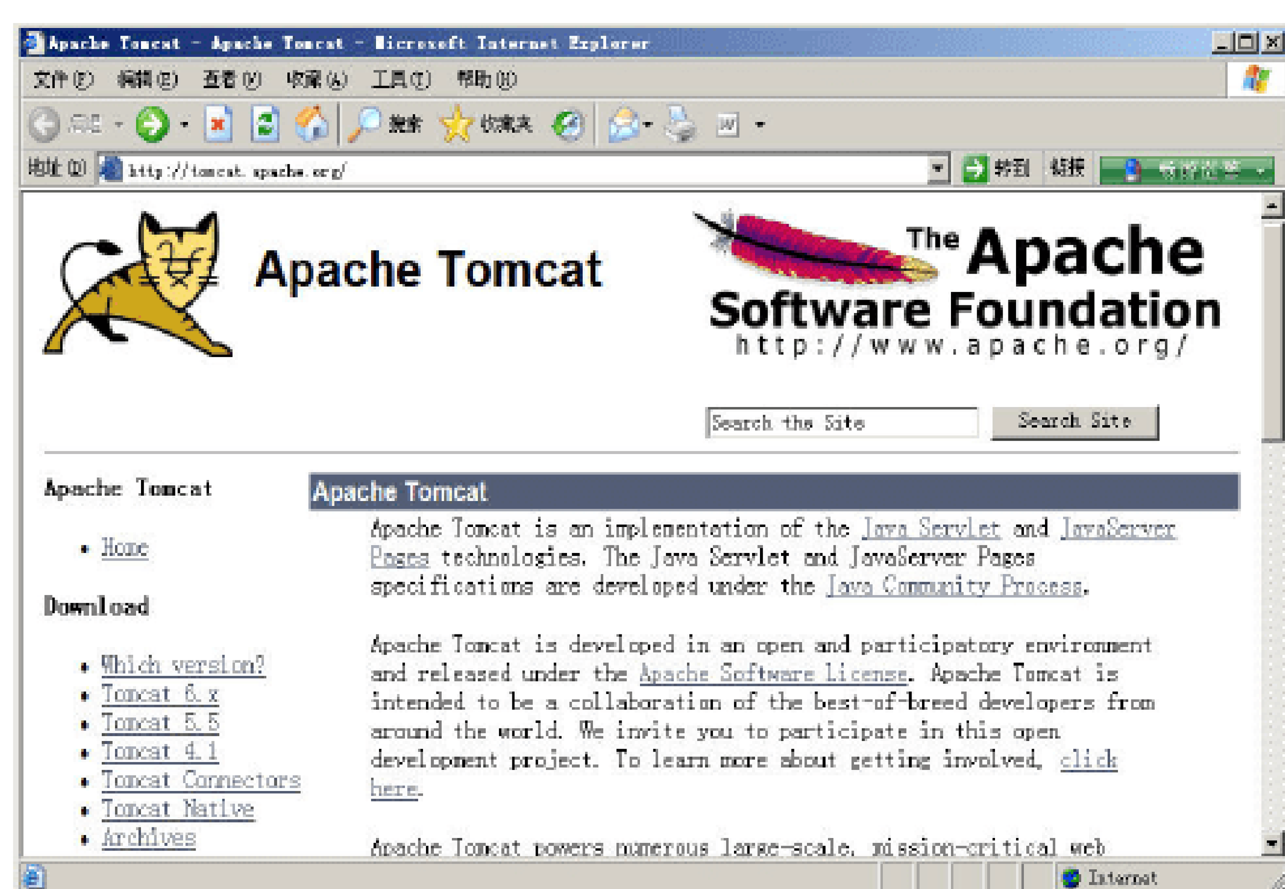


图 1.20　Tomcat 下载首页

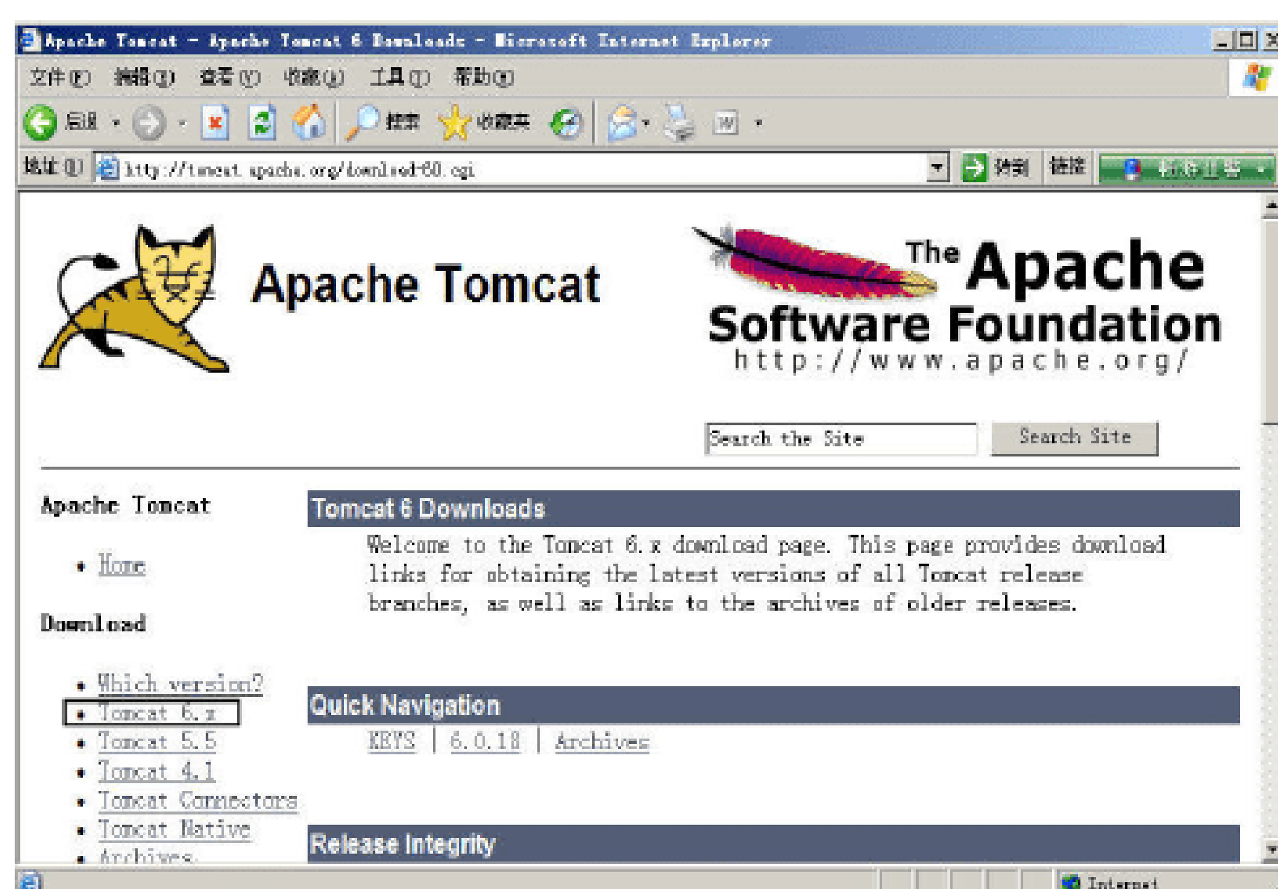


图 1.21　Tomcat 下载页面

（3）选择下载页面中 Binary Distributions 栏目下 Core 选项中的任何一项，建议下载 Windows Service Installer 的 exe 安装文件，如图 1.22 所示。

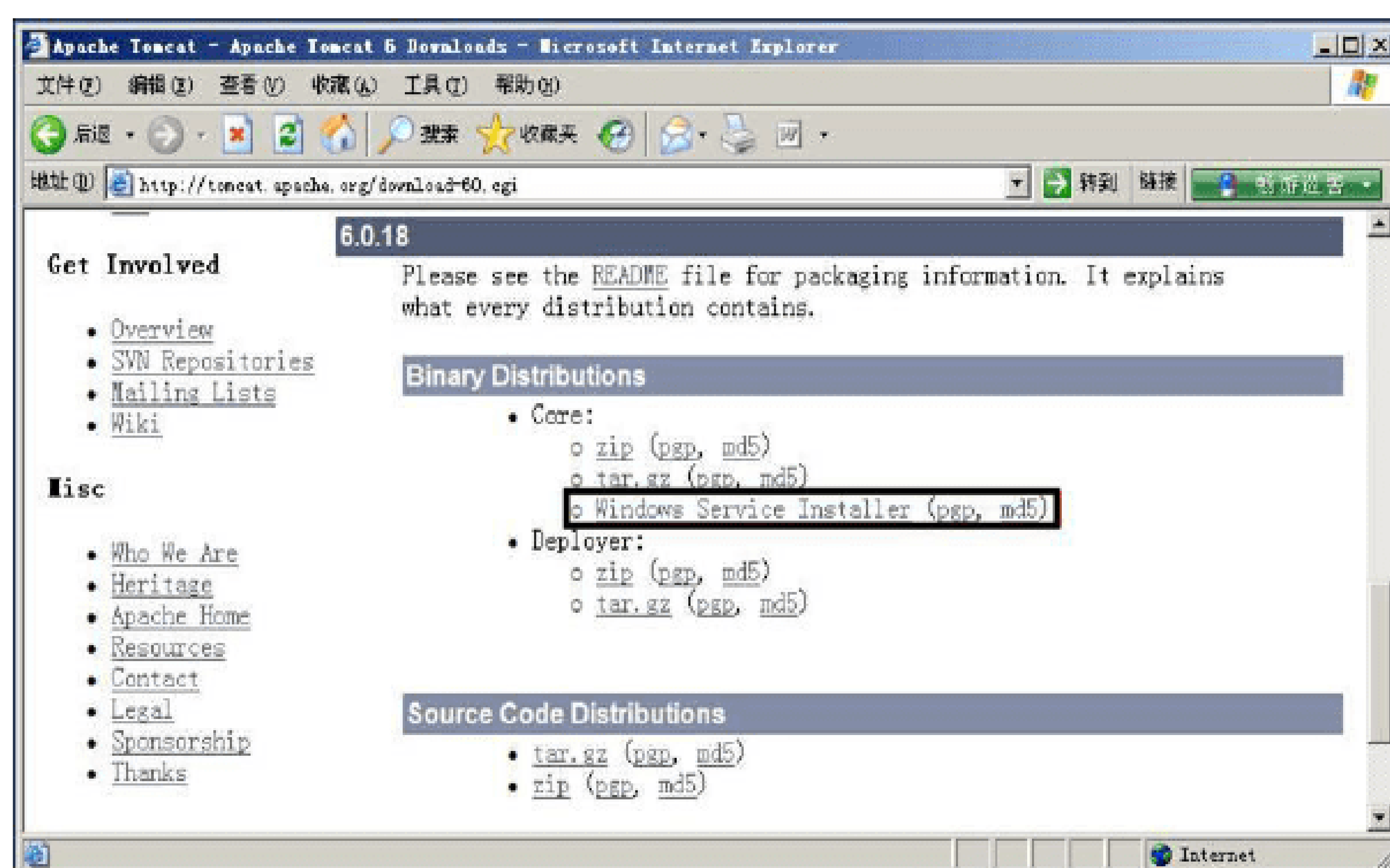


图 1.22　选择 Tomcat 类型

下载完 Tomcat 后，就可以安装该服务器，具体步骤参考 1.2.4 节。

1.2.4　安装服务器 Tomcat

在安装 Tomcat 之前必须安装 JDK，因为在安装过程中该服务器要自动查找 JDK 的目录位置。在具体安装时，还要注意 Tomcat 和 JDK 这两个软件版本的限制。具体安装步骤如下。

（1）双击 Tomcat 安装程序（apache-tomcat-6.0.18.exe），接着就会通过 Windows Installer 开始安装过程，如图 1.23 所示。

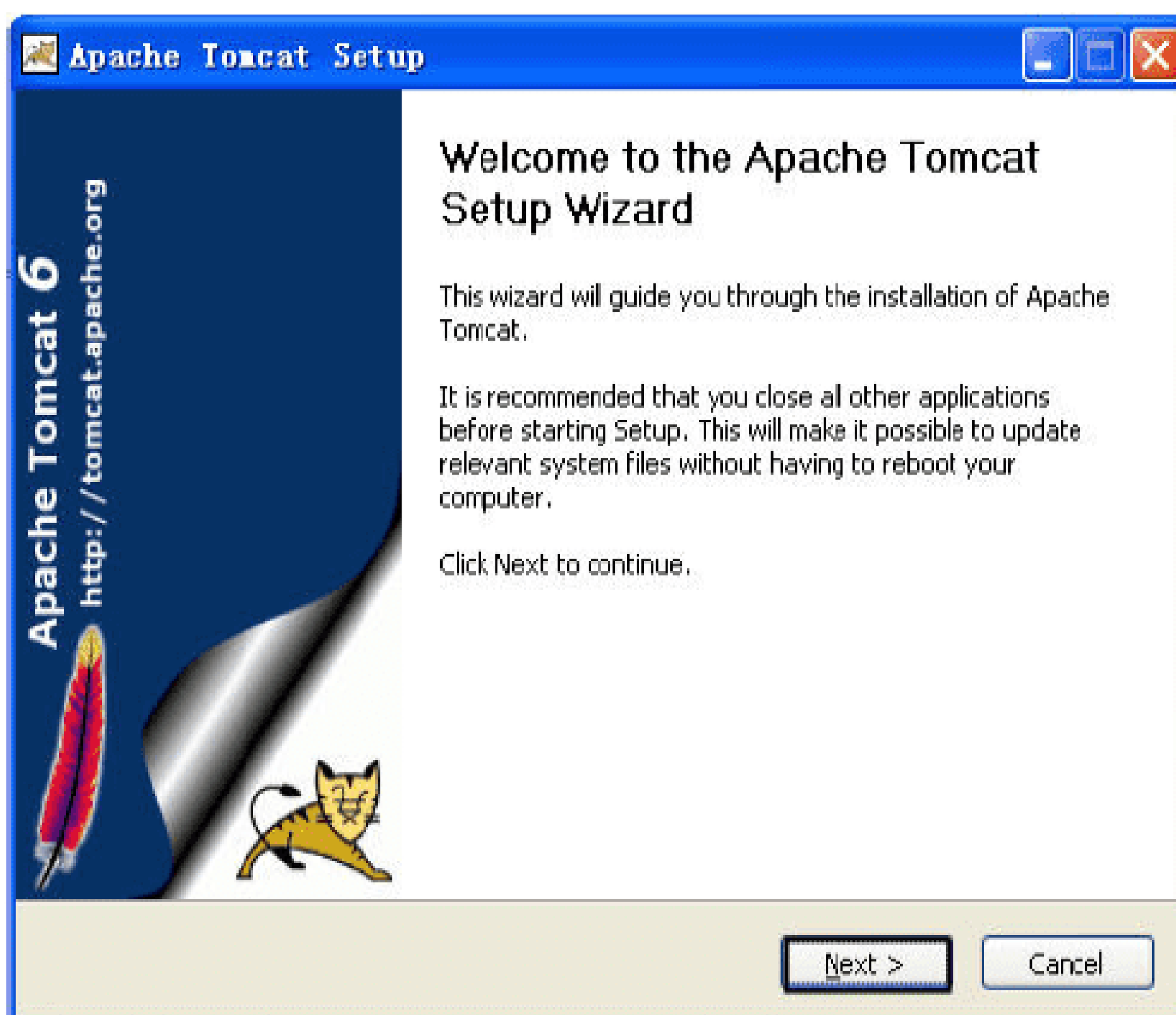


图 1.23　Tomcat 欢迎界面

（2）单击 Next 按钮后，先仔细阅读许可证协议，然后单击 I Agree 按钮，就会进入如图 1.24 所示的 Choose Components（自定义安装）对话框。在该对话框中可以进行安装内容的选择。默认的安装内容如下。

- Tomcat：服务器 Tomcat 的主要组件；
- Start Menu Items：在开始菜单里增加管理 Tomcat 的快捷方式；
- Documentation：Tomcat 的技术文档；
- Examples：Web 应用程序的例子。

（3）单击 Next 按钮，就会进入 Choose Install Location（自定义安装路径）对话框，如图 1.25 所示，在该对话框中可以进行安装路径的选择。

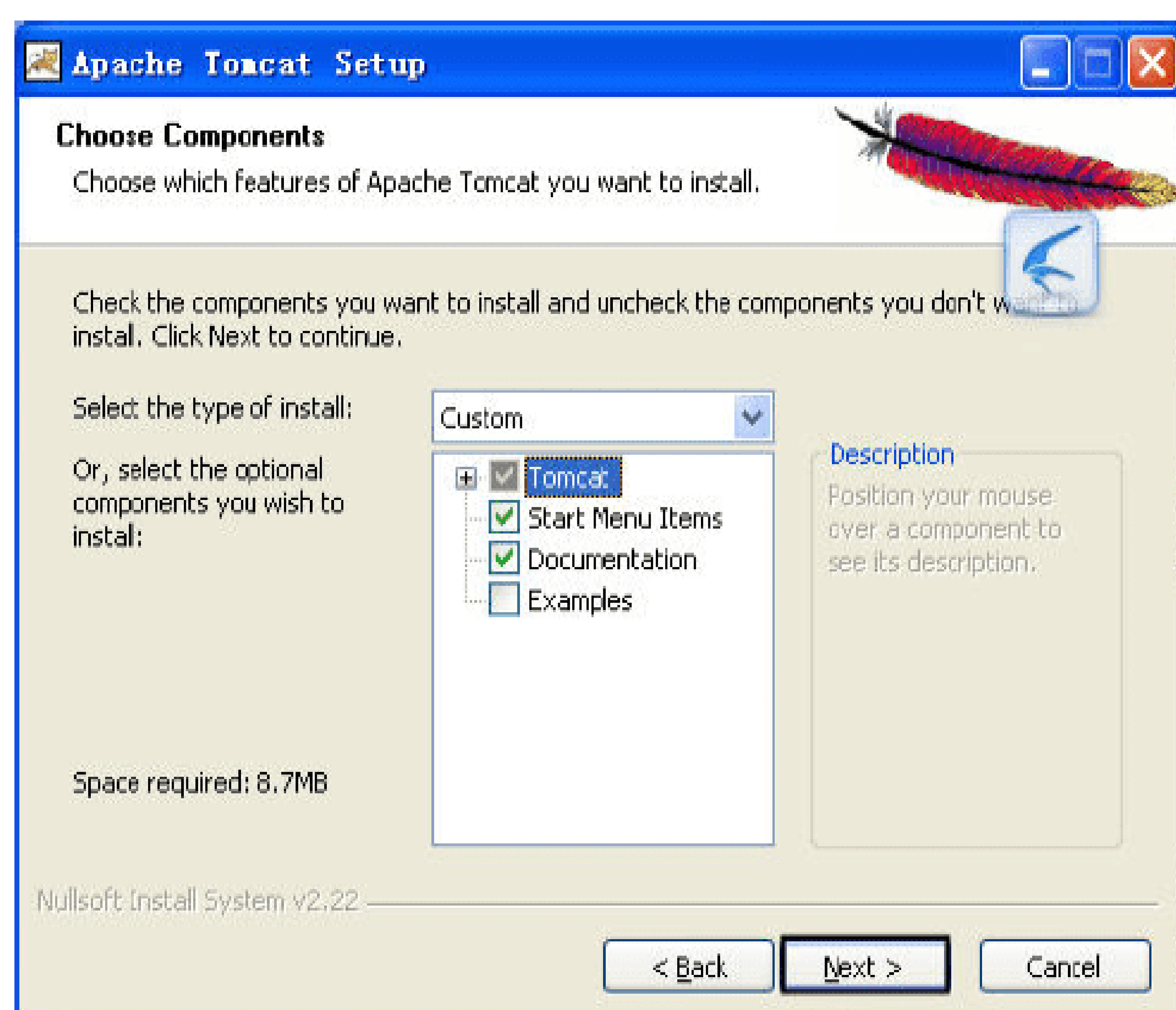


图 1.24　选择安装内容

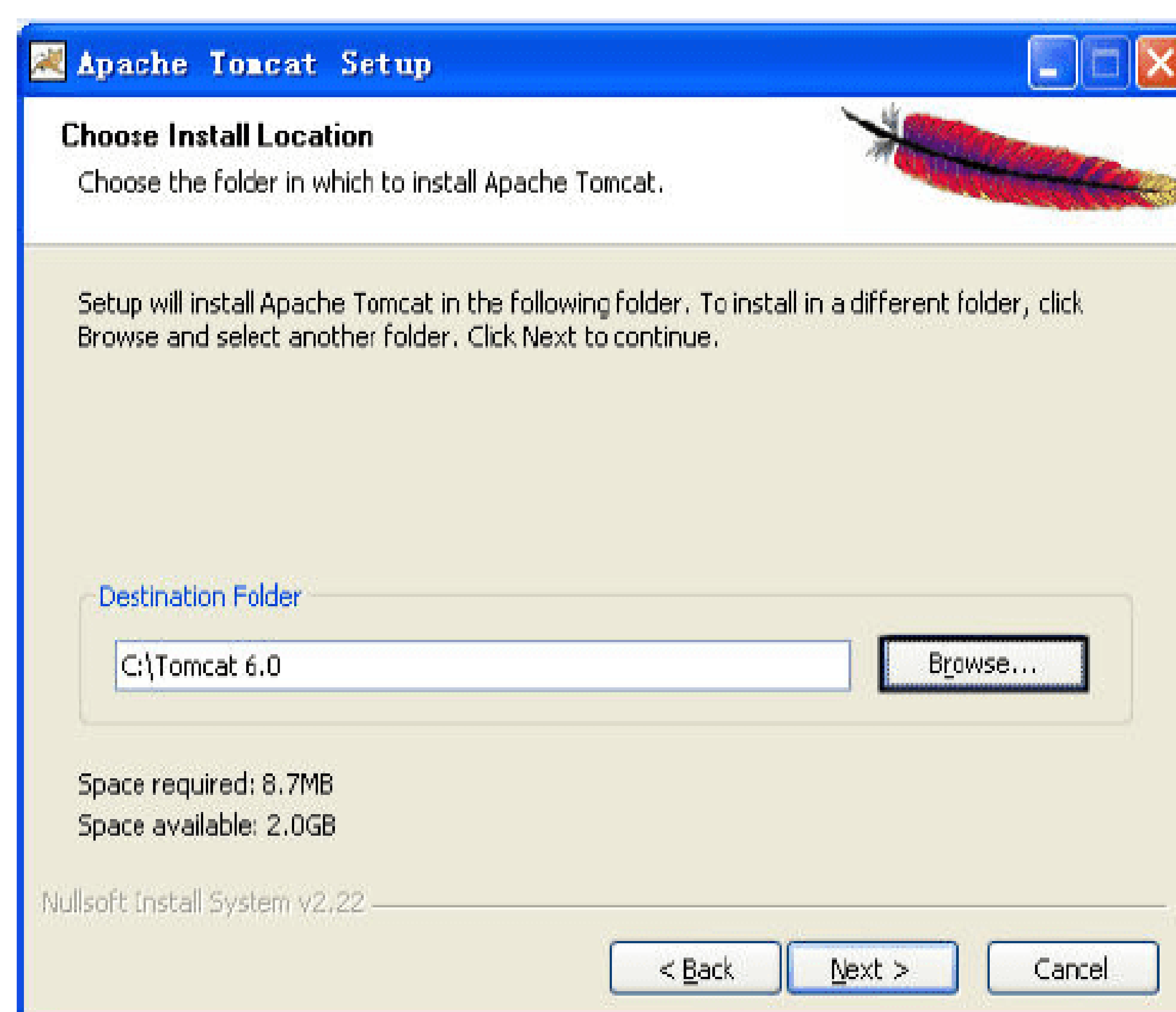


图 1.25　更改安装路径

一般推荐路径是“C:\Tomcat 6.0”，所以需要更改默认安装路径。单击 Browse 按钮，然后在“浏览文件夹”对话框中选择相对应的路径。

（4）单击 Next 按钮，就会进入 Configuration（基本配置）对话框，如图 1.26 所示。在该对话框中可以进行最基本的配置。

- HTTP/1.1Connector Port：用来设置 Tomcat 的端口号，默认为 8080；
- Administrator Login：用来设置登录用户的用户名（User Name）和密码（Password）。

（5）单击 Next 按钮，就会进入 Java Virtual Machine（Java 虚拟机）配置对话框（如图 1.27 所示），在该对话框中可以进行 Java 虚拟机的配置。该项配置一般会自动寻找虚拟机的目录位置，用户只需要确认即可。

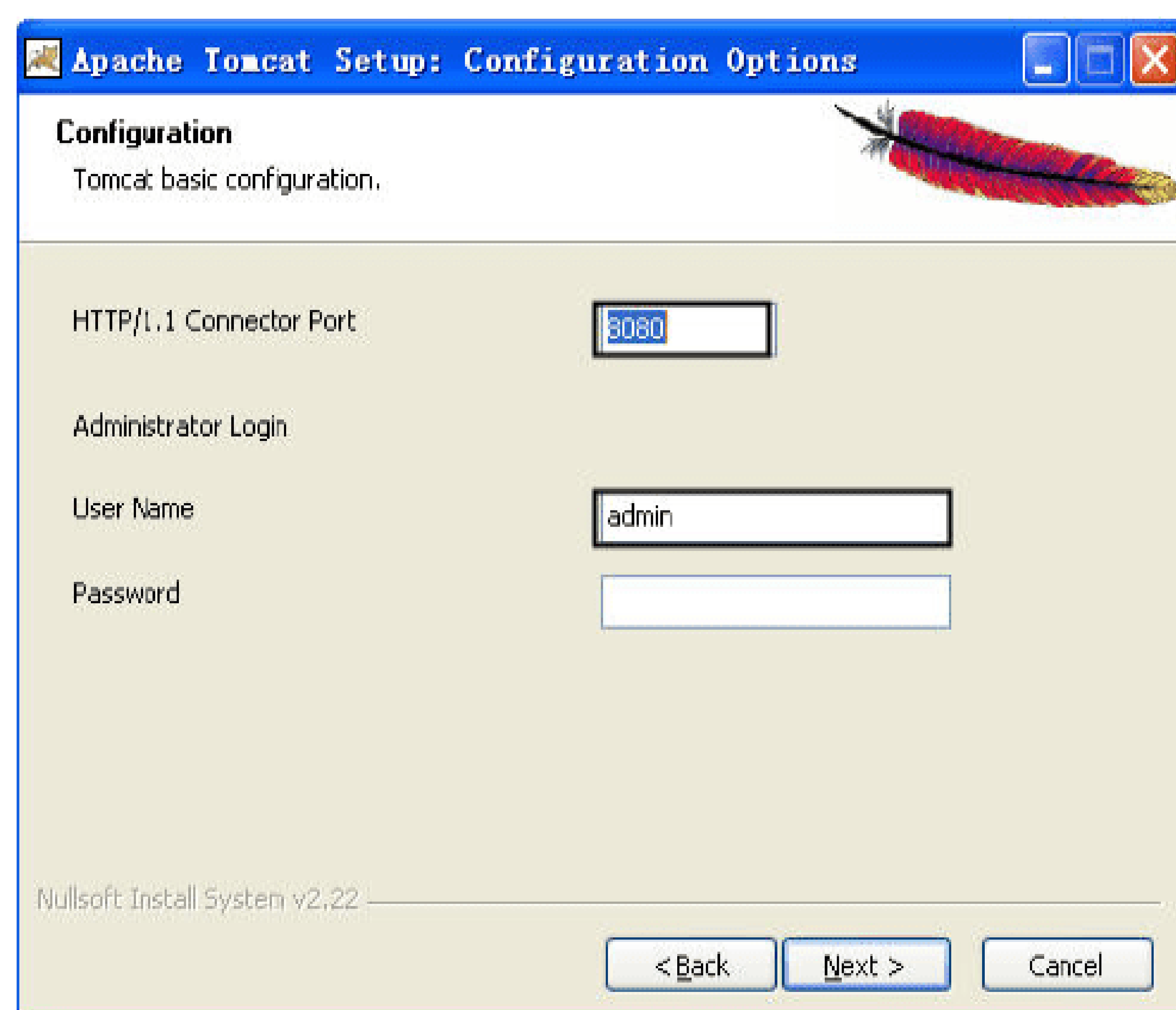


图 1.26　配置 Tomcat

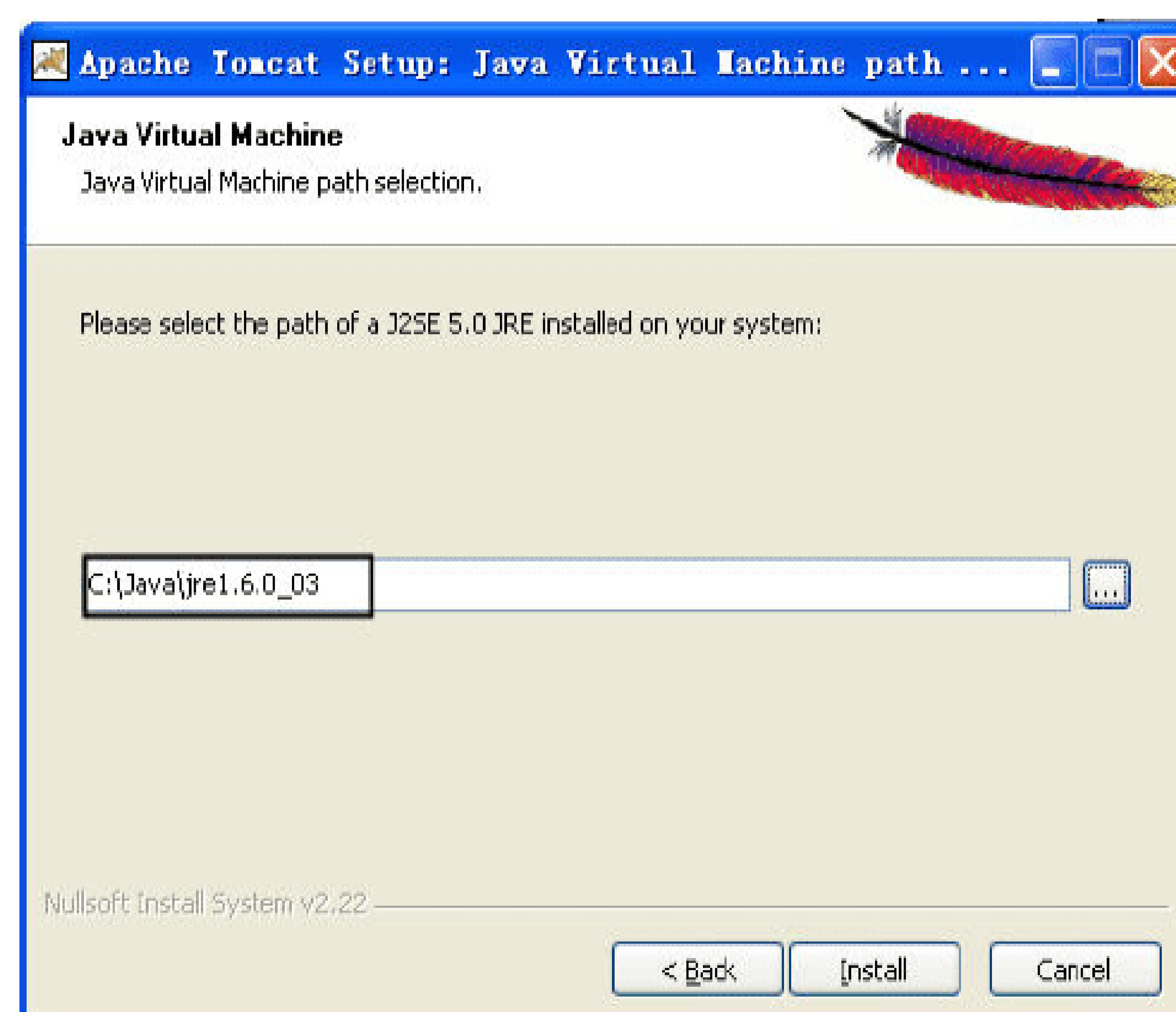


图 1.27　确认 JVM 路径

（6）最后，单击 Install 按钮就会自动完成服务器 Tomcat 的安装。

1.2.5　下载开发环境 MyEclipse

MyEclipse 是由 Genuitec 公司开发的一款商业软件，从本质上讲，它是基于 Eclipse 的 Java EE 方面的插件。该软件除了支持代码编写、编译和测试等，还增加了 UML 双向建模工具、JSP/Srutsdesigner、可视化的 Hibernate/ORM 工具、Spring 和 Web Services 等各个方面的功能。目前最新的版本为 MyEclipse 6.6，可以通过下面的步骤来实现该平台的下载。

（1）首先访问下载 MyEclipse 的官方网站（http://myeclipseide.com/），如图 1.28 所示。



图 1.28　MyEclipse 下载首页

（2）打开 MyEclipse 首页后，单击 Try Now 按钮，进入下载页面。接受协议后，然后单击 MyEclipse 6.6 版本上的 DOWNLOAD 按钮，才真正进入下载页面。

（3）在下载页面中建议下载 ALL In One 版本，因为该版本无须另外下载安装和配置 JDK、Eclipse 3.3，是快速安装 MyEclipse 的最佳选择。

下载完 MyEclipse 6.6 后，就可以安装该开发工具。

1.2.6　安装开发环境 MyEclipse

不同类型的 MyEclipse 安装过程不一样，“ALL In One”类型的 MyEclipse 安装过程很简单，只要双击安装程序就可以，不需要安装者进行复杂详细设置。具体安装步骤如下。

（1）双击 MyEclipse 6.6 安装程序（MyEclipse_6.6.0_E3.3.1_Installer.exe），接着就会通过 Windows Installer 开始安装过程，如图 1.29 所示。

（2）单击 Next 按钮后，在出现的对话框中首先仔细阅读许可证协议，然后选择接受该协议。接着在出现的 Choose Destination Location（路径选择）对话框中更改安装路径为“C:\MyEclipse6.6”。最后单击 Install 按钮，就可以自动安装该软件，如图 1.30 所示。

（3）安装完后，就可以通过开始菜单上的 MyEclipse 6.6 选项来启动 MyEclipse 6.6，运行界面如图 1.31 所示。

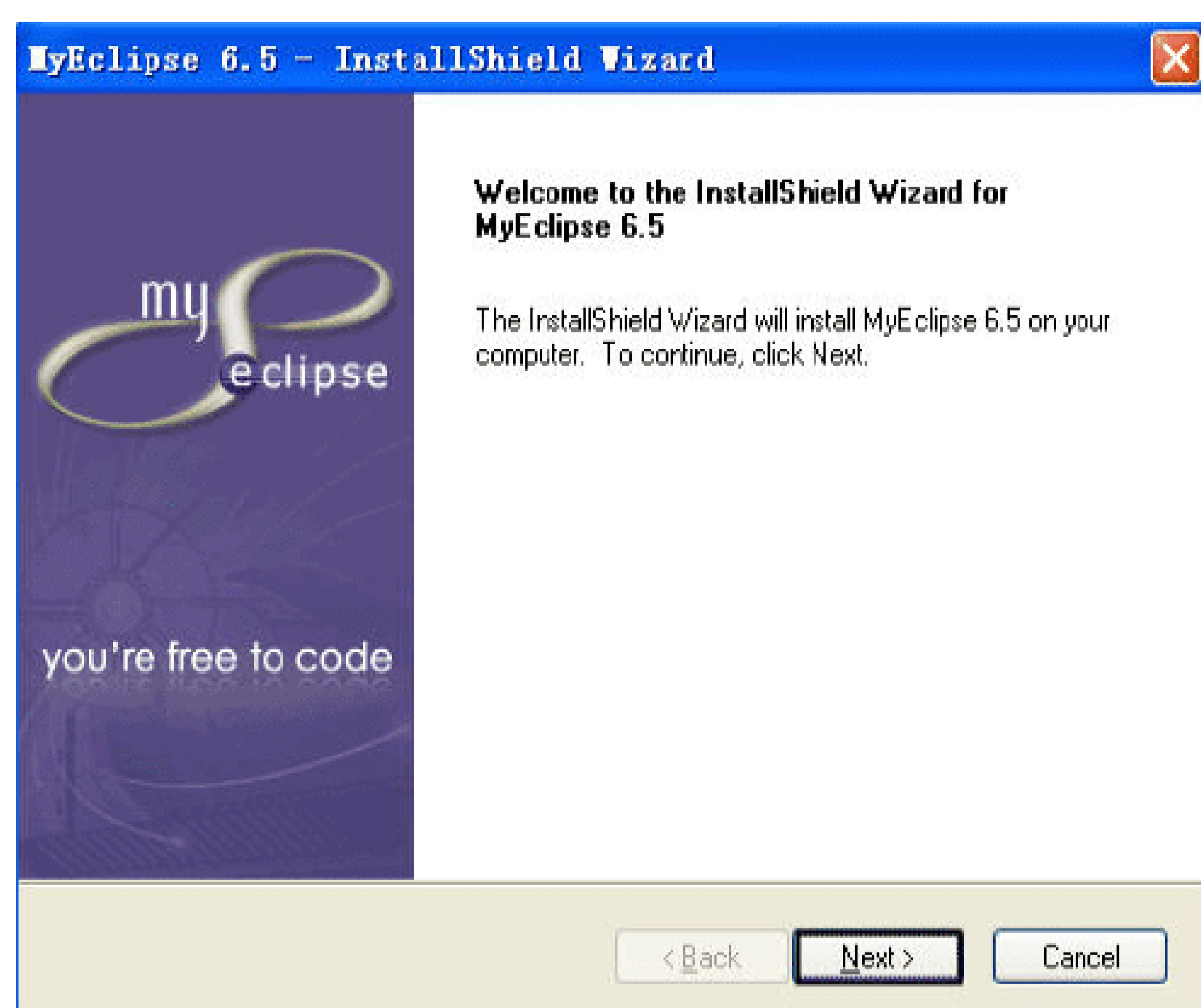


图 1.29　MyEclipse 欢迎界面

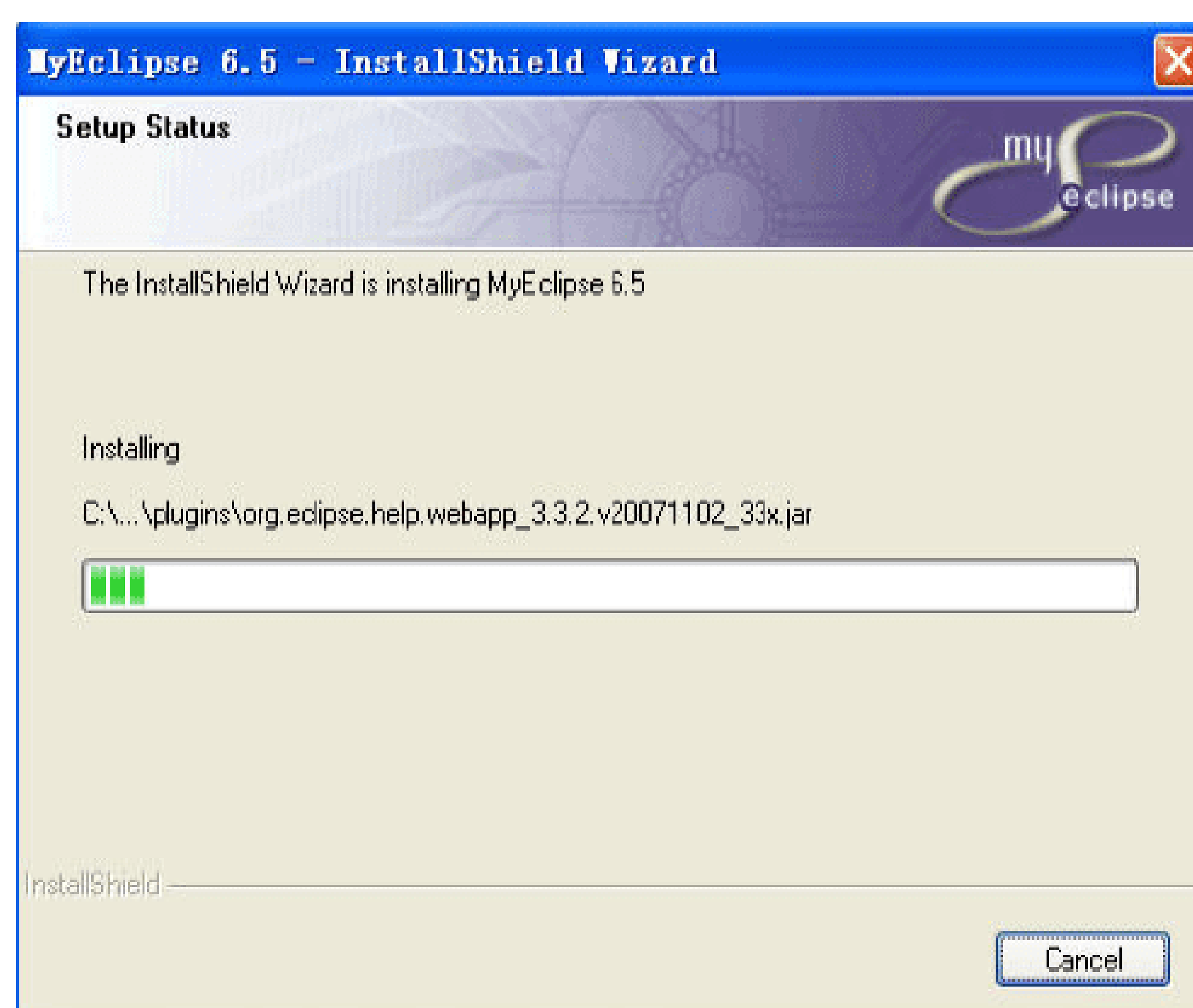


图 1.30　安装 MyEclipse

（4）如果使用者购买或者有 MyEclipse 6.6 的注册码，可以选择 MyEclipse 菜单下的 Subscription Information 命令，在出现如图 1.32 所示的对话框中把相应的信息填写到对应的位置就可以。

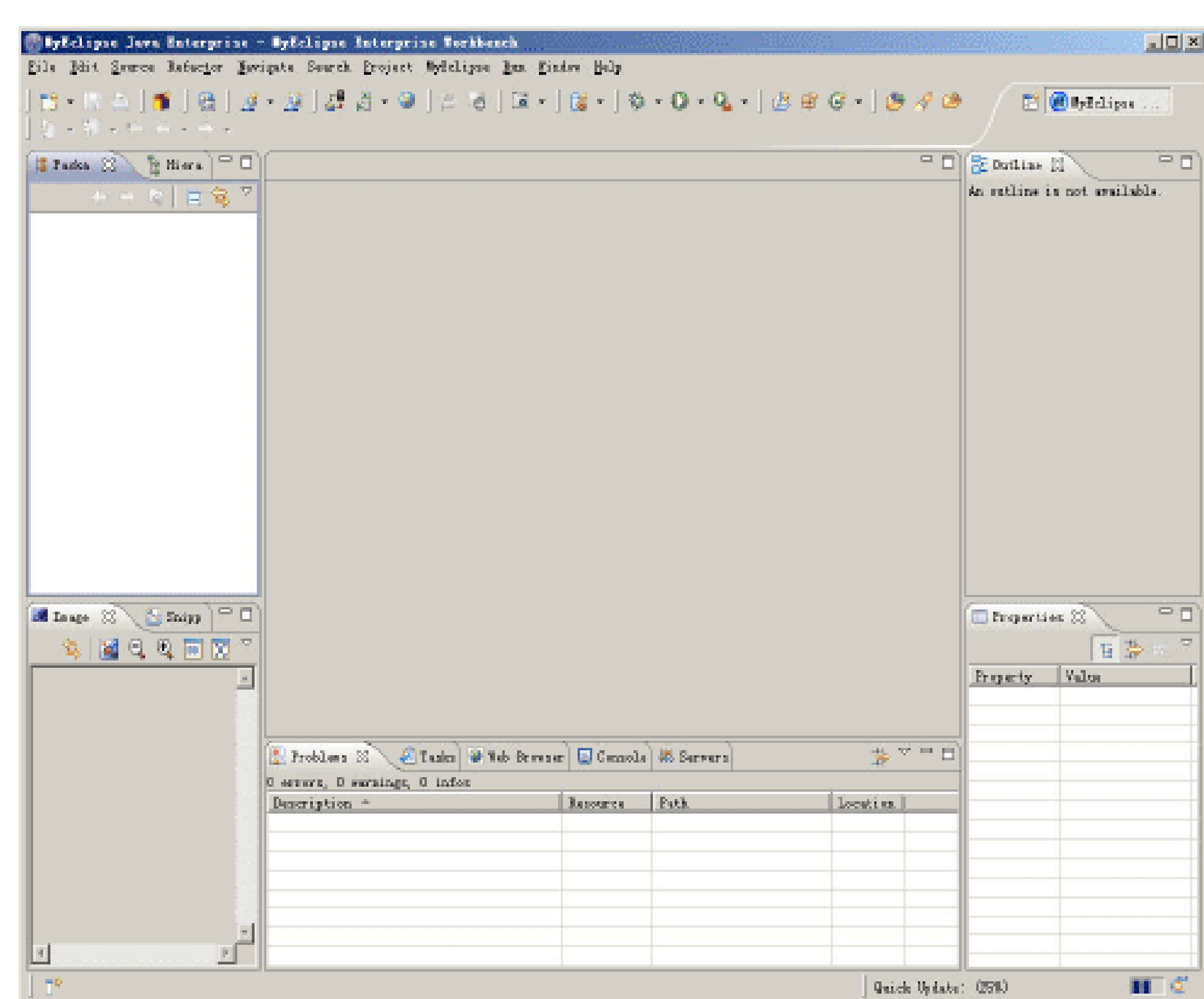


图 1.31　MyEclipse 运行界面

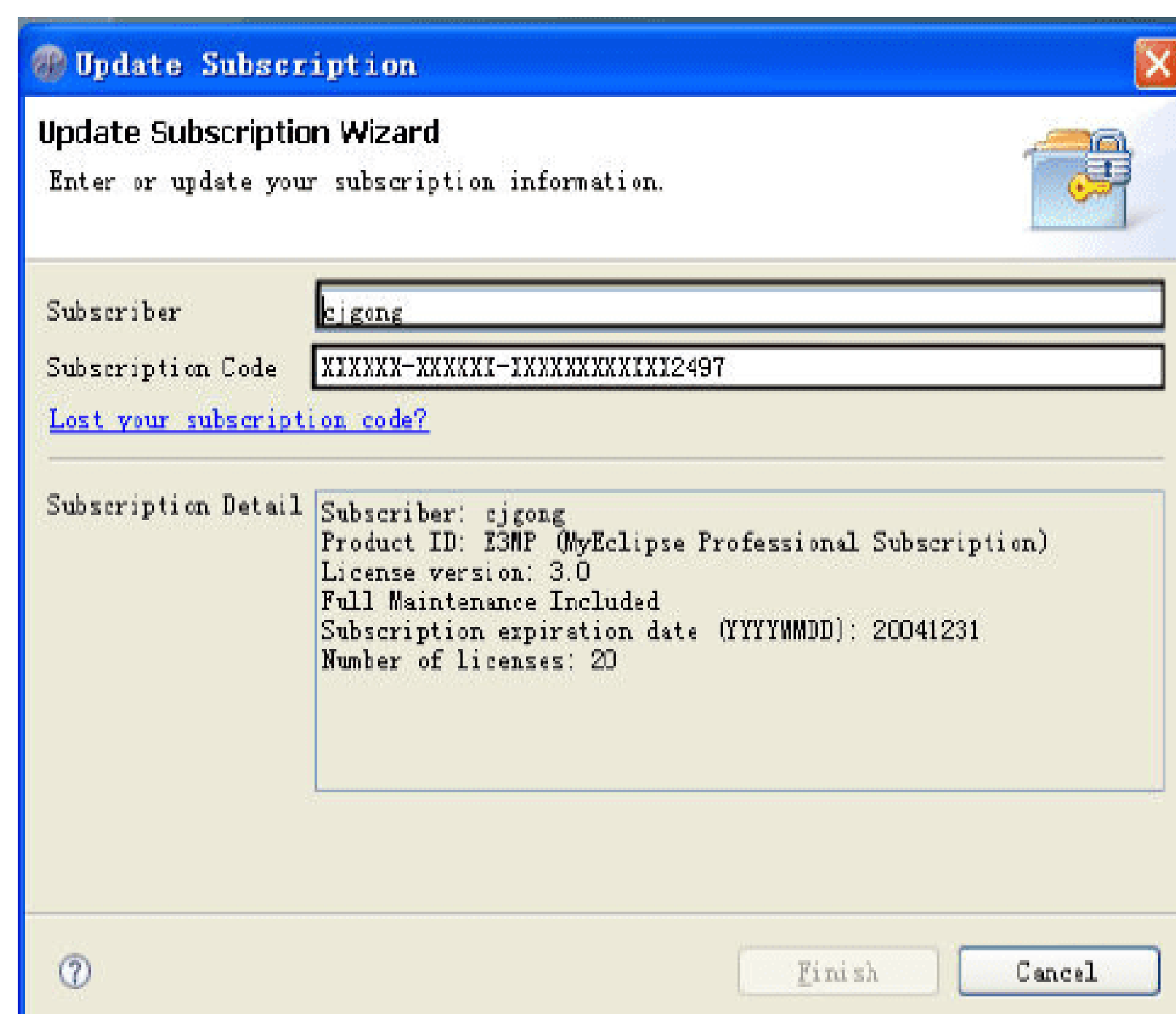


图 1.32　注册 MyEclipse

1.2.7　下载数据库服务器 MySQL

MySQL 是完全网络化的跨平台关系型数据库系统，具有了客户机/服务器体系结构的分布式数据库管理系统。该软件不仅具有功能强、使用简便、管理方便、运行速度快、安全可靠性强等优点，而且还可利用许多语言编写访问 MySQL 数据库的程序。

目前最新的版本为 mysql-5.1.19-rc-win31，可以通过下面的步骤实现该平台的下载，具体如下。

（1）首先访问下载 MySQL 的官方网站（http://dev.mysql.com/），如图 1.33 所示。

（2）打开 MySQL 首页后，单击页面的 Download 导航栏，就会进入下载页面，如图 1.34 所示。

（3）在下载页面中选择 MySOL 5.0 超级链接，就会打开关于 MySOL 5.0 平台的下载

页面，如图 1.35 所示。选择 Windows 平台的类型，单击 Pick a mirror 超级链接开始下载 MySOL 5.0。



图 1.33　MySQL 下载首页



图 1.34　选择 MySQL 版本

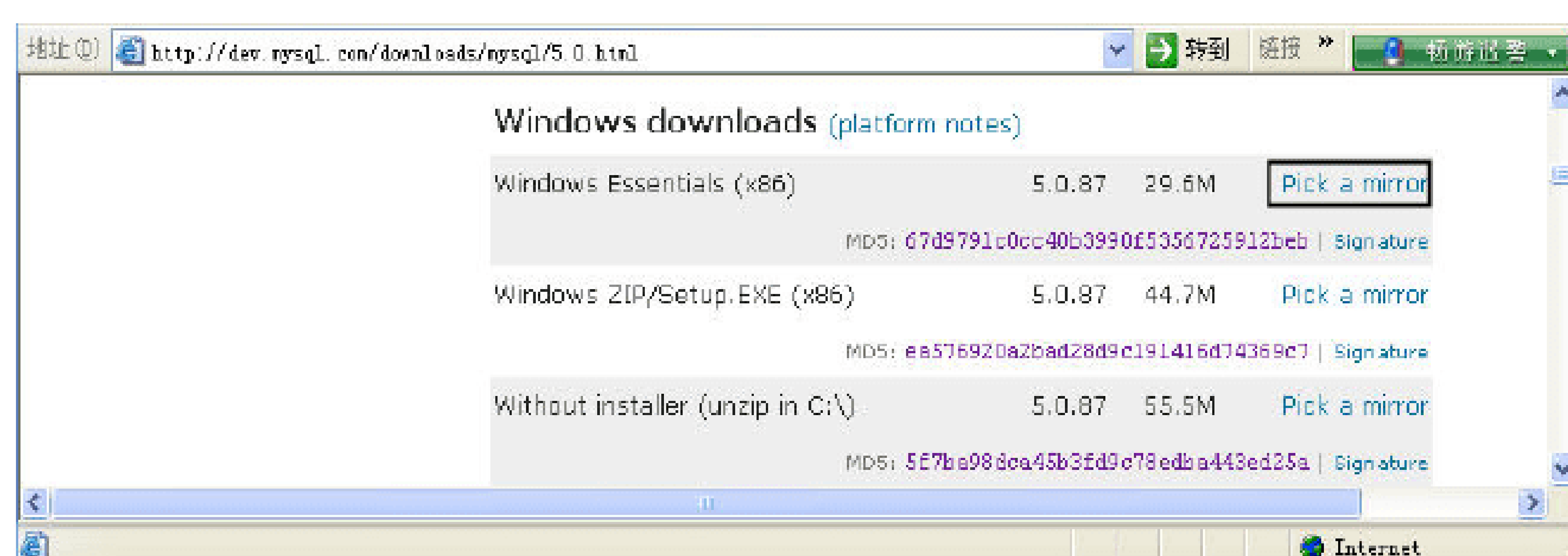


图 1.35　选择 MySQL 5.0 平台

下载完 MySOL 5.0 后，就可以安装该数据库。

1.2.8　安装数据库服务器 MySOL

1.2.7 节介绍了如何下载数据库 MySOL，下载完数据库 MySOL 安装程序后就可以开始安装该数据库。具体的安装步骤如下。

（1）双击 MySOL 安装程序（mysql-5.1.19-rc-win31.exe），接着就会使用 Windows Installer 开始安装过程，如图 1.36 所示。

（2）单击 Next 按钮后，在弹出的对话框中（如图 1.37 所示）就可以进行安装类型的选择。

默认的安装类型如下。

- Typical：默认的安装类型；
- Complete：完全的安装类型；
- Custom：自定义的安装类型。



图 1.36　MySOL 欢迎界面

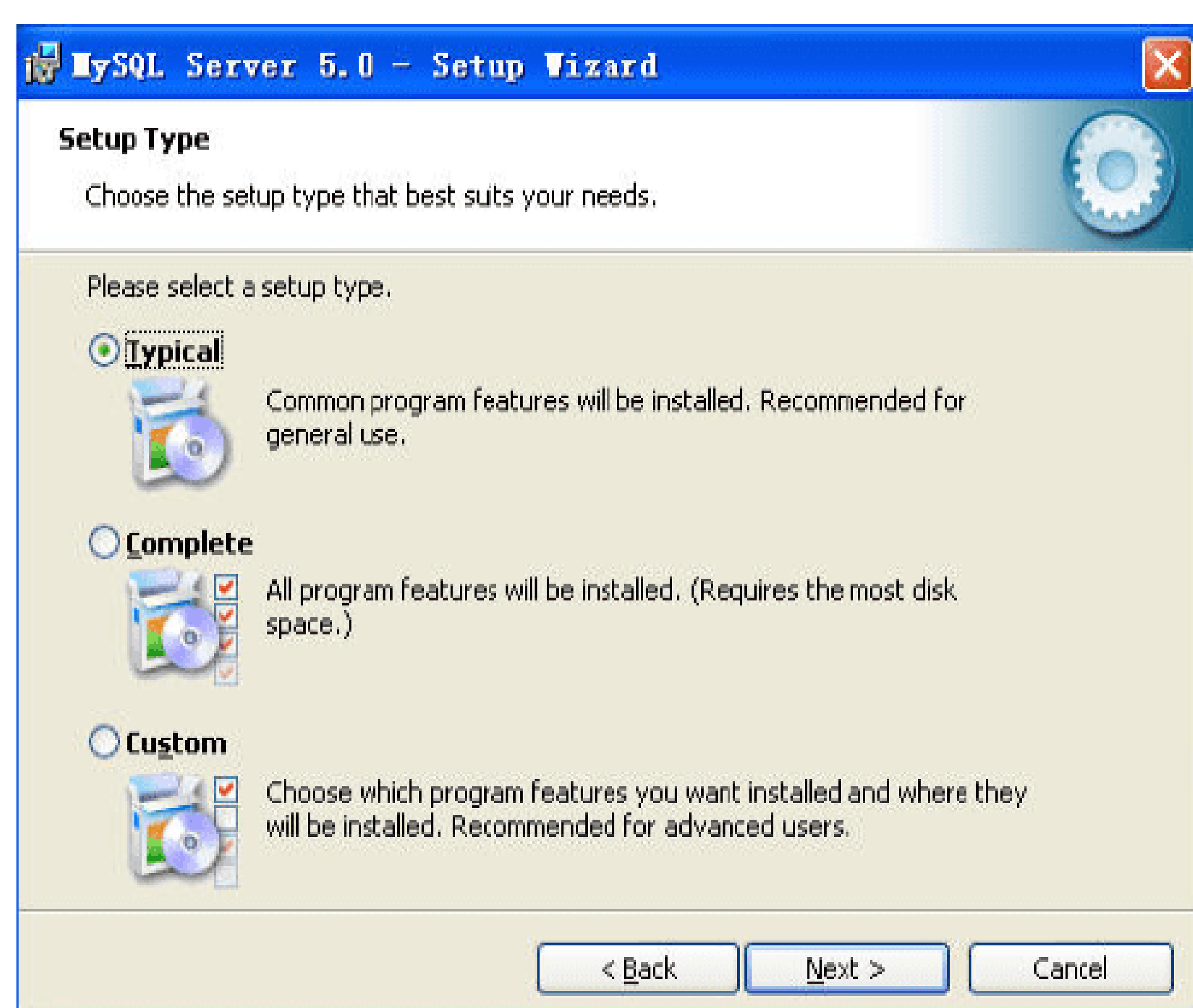


图 1.37　MySOL 安装类型对话框

（3）选择 Typical 单选按钮，单击 Next 按钮，就会出现如图 1.38 所示的对话框。在该对话框中确认一下安装的信息，单击 Install 按钮开始对 MySOL 软件进行安装。安装完后在弹出的对话框中会询问是否现在进行配置，如图 1.39 所示。如果不想现在配置，可以取消选中 Configure the MySQL Server now 复选框，然后单击 Finish 按钮就会完成对 MySOL 软件的安装。

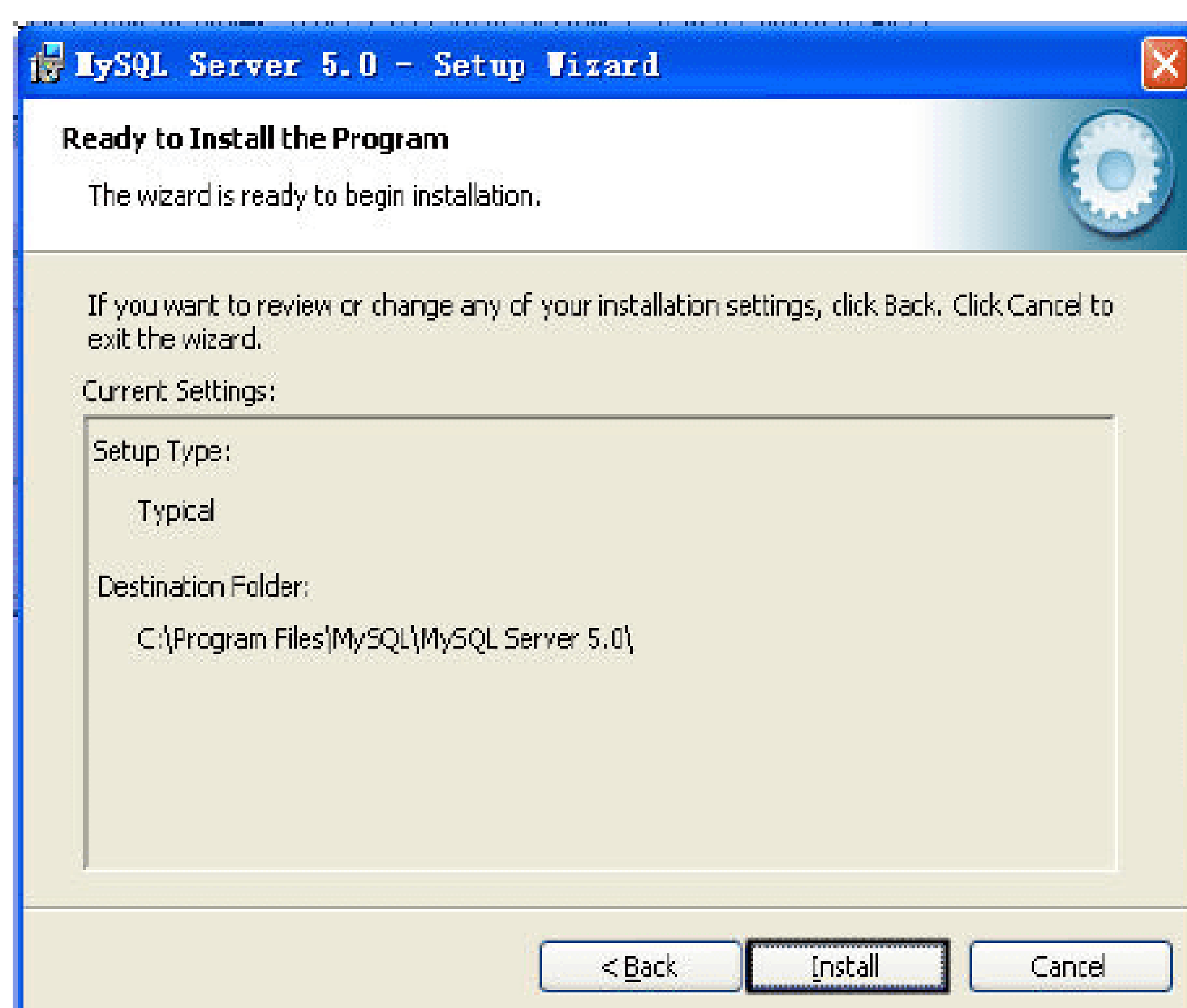


图 1.38　安装信息的确认

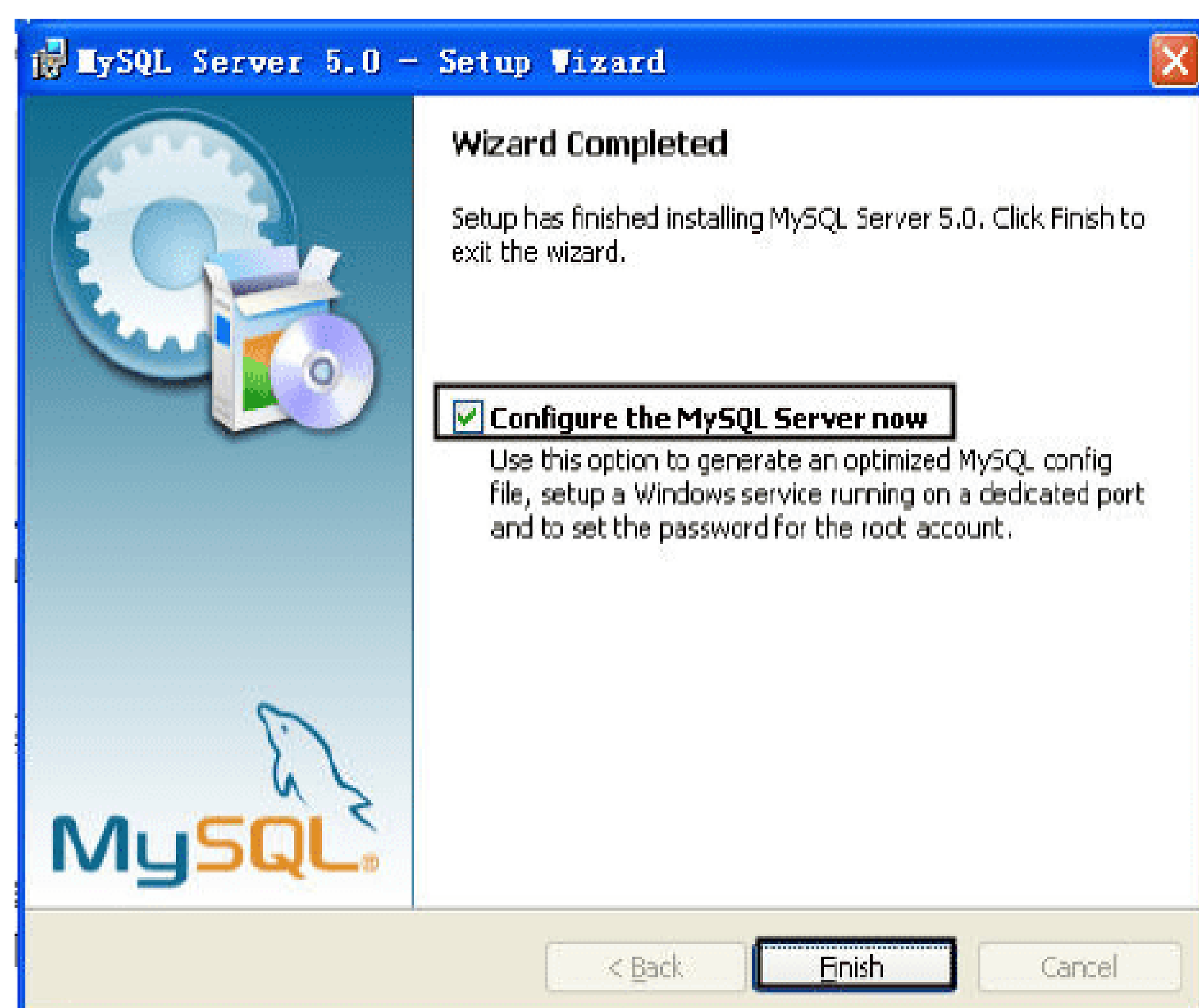


图 1.39　完成 MySOL 安装

1.2.9　下载数据库服务器 Oracle

Oracle 是中国殷墟（Yin Xu）出土的甲骨文（oracle bone inscriptions）英文翻译的第一个单词，在英语中是“神谕”的意思。目前比较稳定的版本为 Oracle Database 10g，可以通过下面的步骤来实现该数据库软件的下载，具体步骤如下。

（1）首先访问下载 Oracle 的官方网站（http://www.oracle.com/index.html），如图 1.40 所示。为了方便下载，可以选择最上面的 Worldwide 链接，然后选择 China 选项，如图 1.41 所示，就会出现中文版本的官方网站。

（2）在如图 1.42 所示的 Oracle 首页中，单击页面右边“下载专区”的“查看所有下载信息”链接，就会进入下载信息页面。

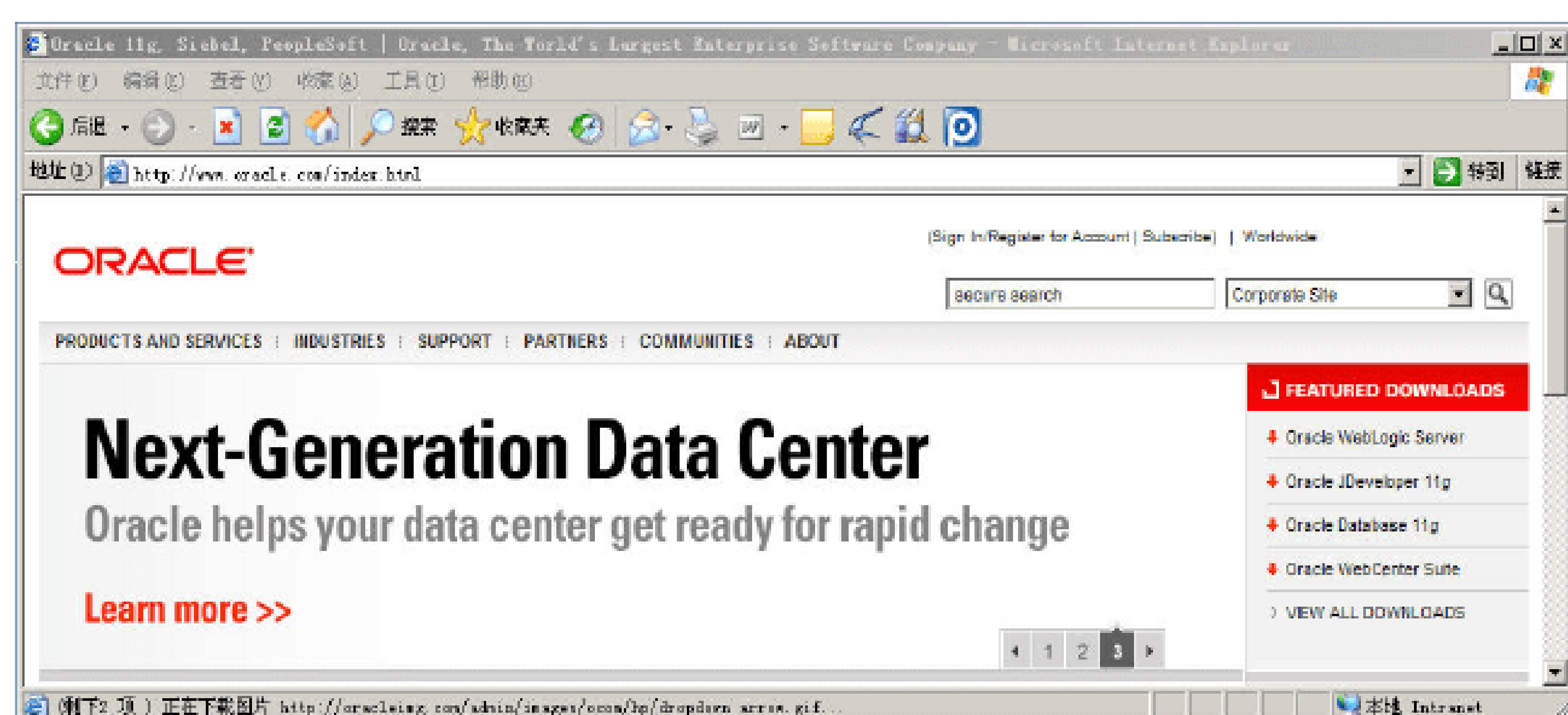


图 1.40　Oracle 首页



图 1.41　改变显示版本

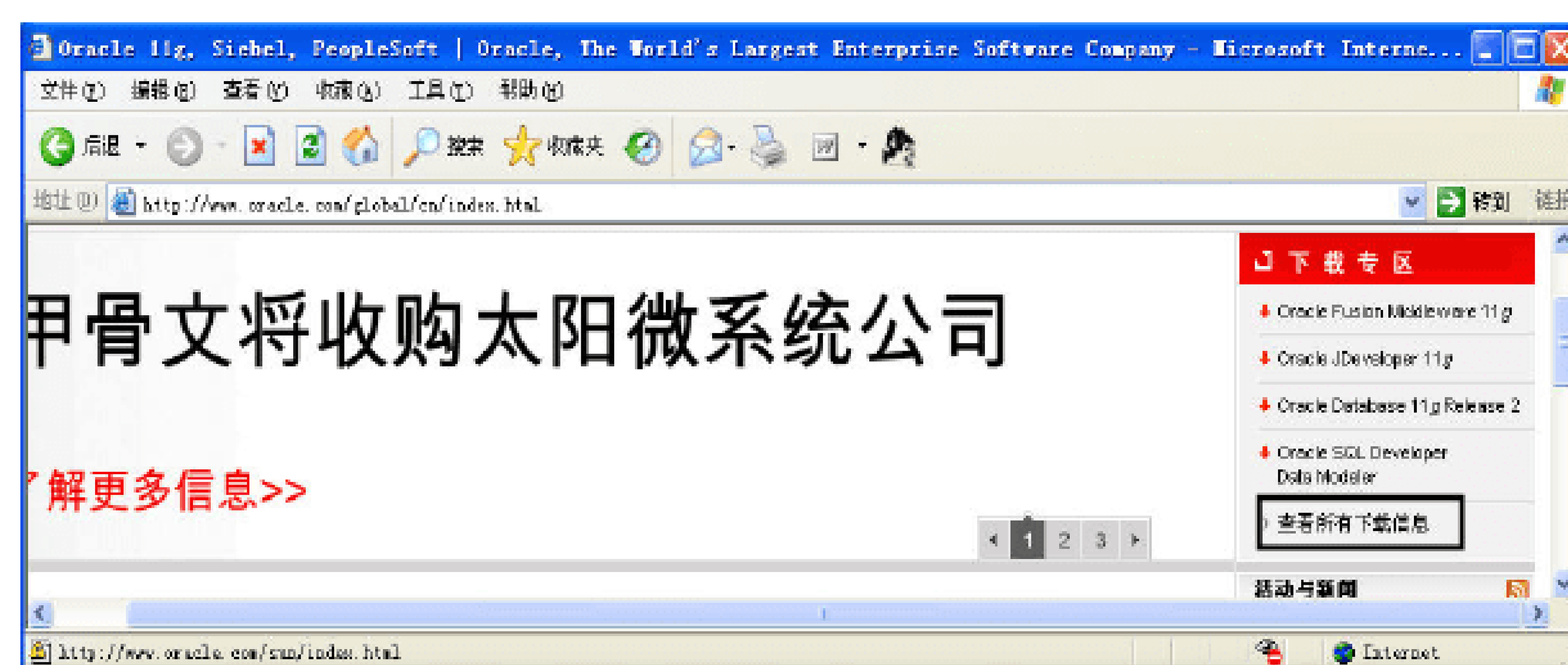


图 1.42　Oracle 中文首页

（3）在如图 1.43 所示的下载信息页面中，单击 Databas 专区中的 Database 10g Express Edition 链接，就可以进入关于 Oracle Database 10g 下载的页面。

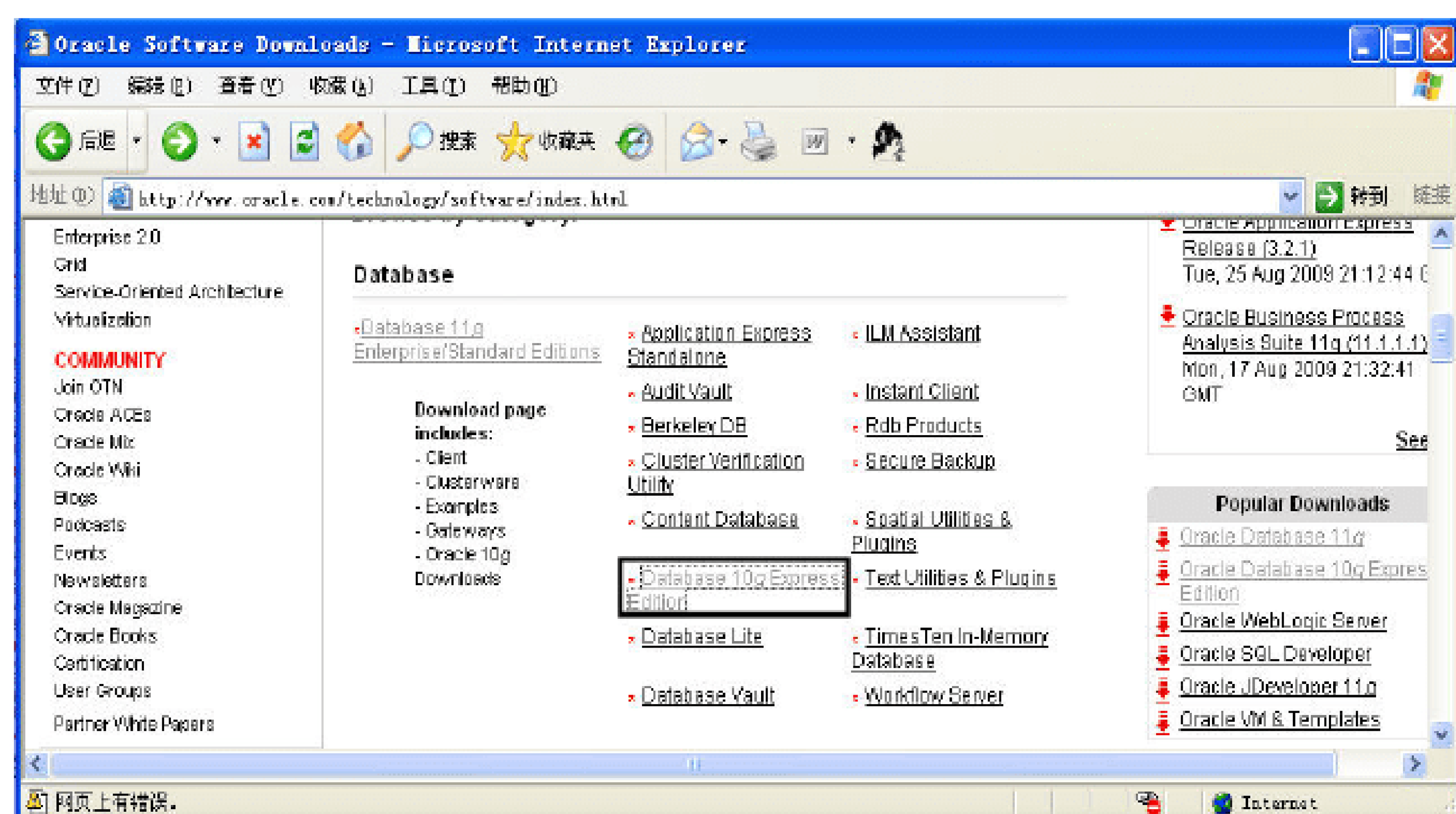


图 1.43　下载信息页面

（4）在如图 1.44 所示的下载页面中，单击 Oracle Database 10g Express Edition for Microsoft Windows 链接，就可以进入关于 Oracle Database 10g 软件的真正下载页面。

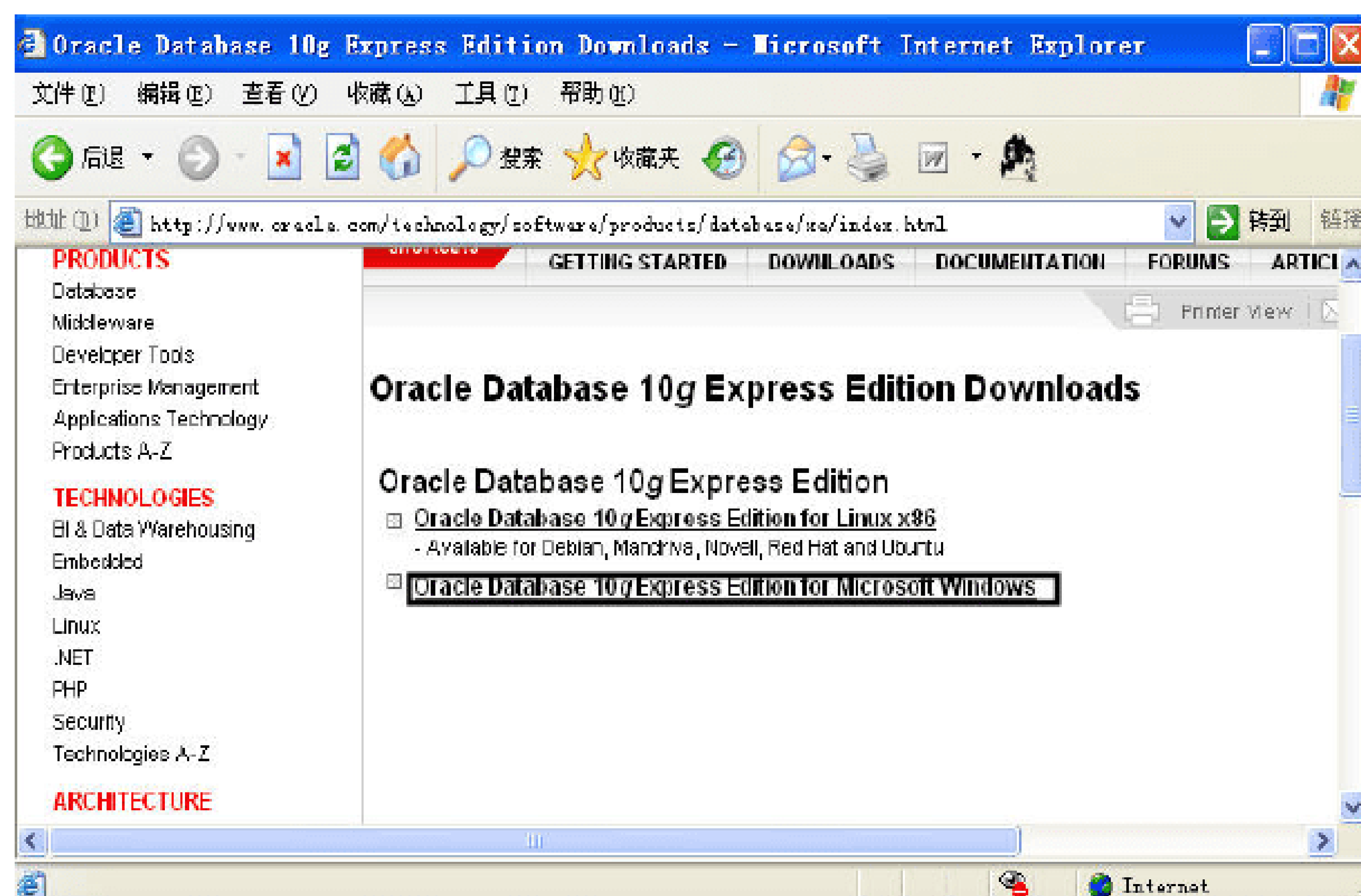


图 1.44　下载页面

（5）在如图 1.45 所示的真正下载页面中，首先接受许可协议，然后选择好所要下载的数据库软件。

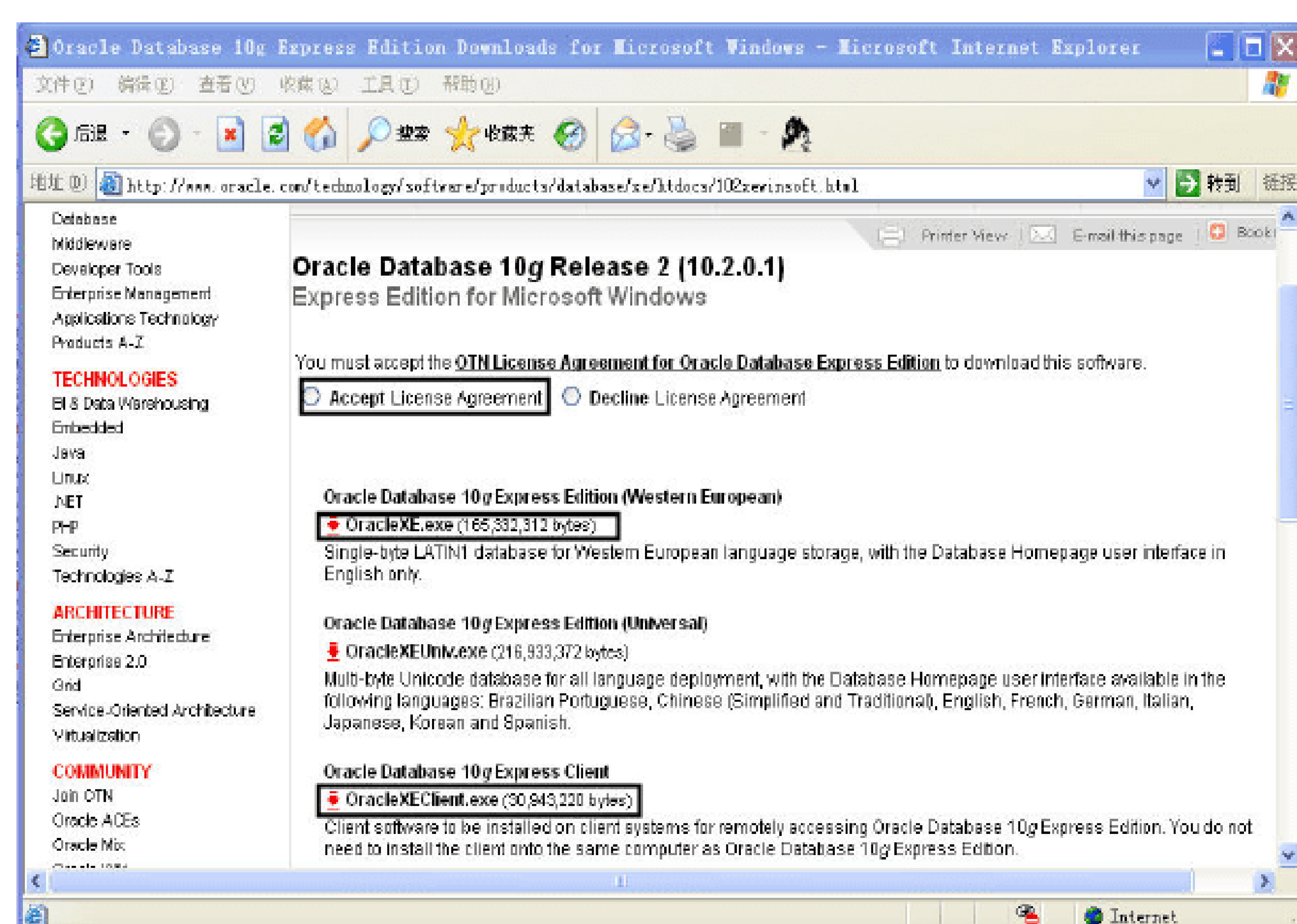


图 1.45　选择相应的数据库软件

下载完 Oracle Database 10g 后，就可以安装该数据库。

1.2.10　安装数据库服务器 Oracle

1.2.9 节介绍了如何下载数据库 Oracle，下载完数据库 Oracle 安装程序后就可以开始安装该数据库。具体的安装步骤如下。

（1）双击 Oracle 安装程序，接着就会使用 Windows Installer 开始安装过程，如图 1.46 所示。

（2）单击“下一步”按钮后，弹出“产品特定的先决条件检查”对话框，如图 1.47 所示。

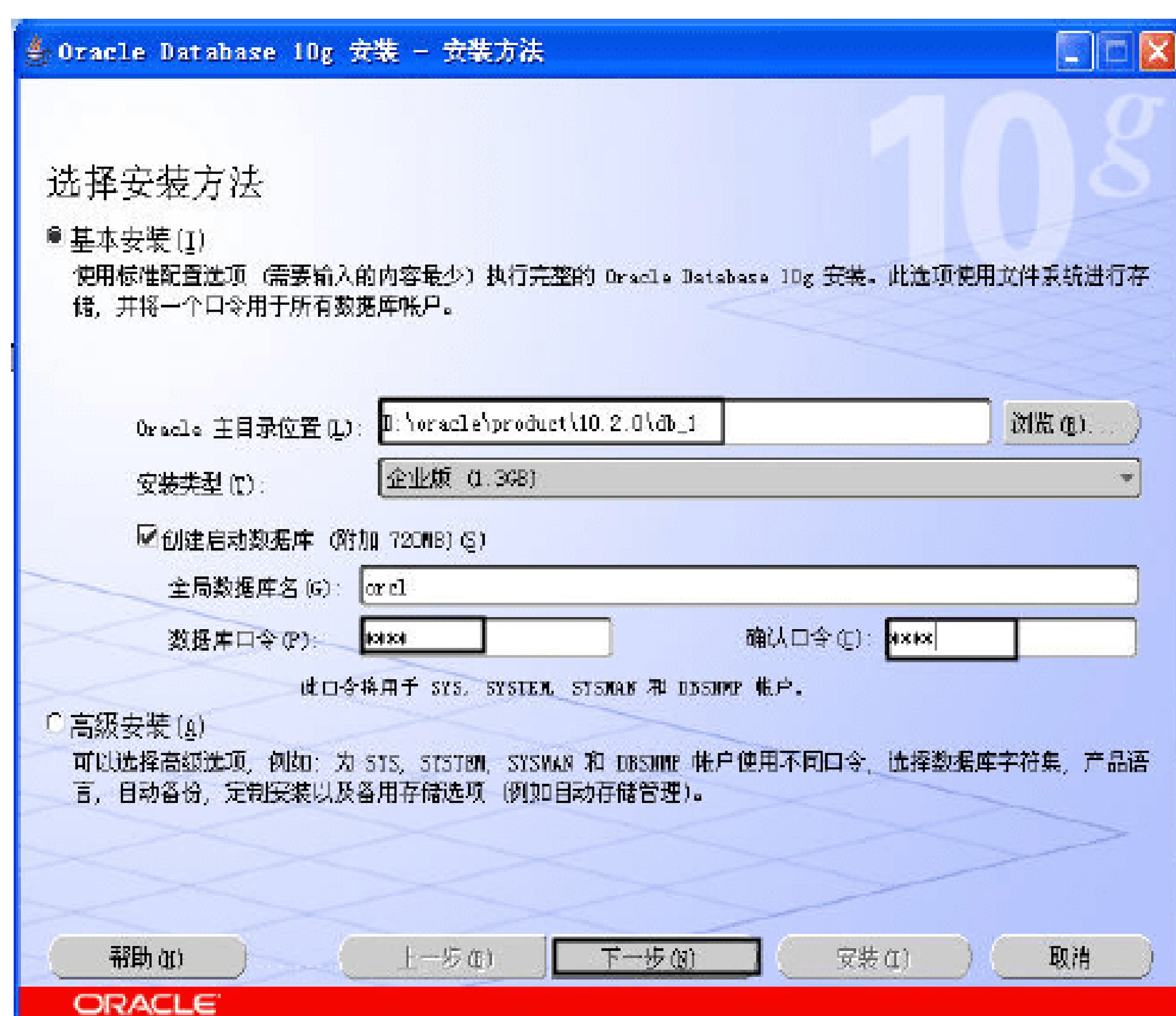


图 1.46　Oracle 欢迎界面

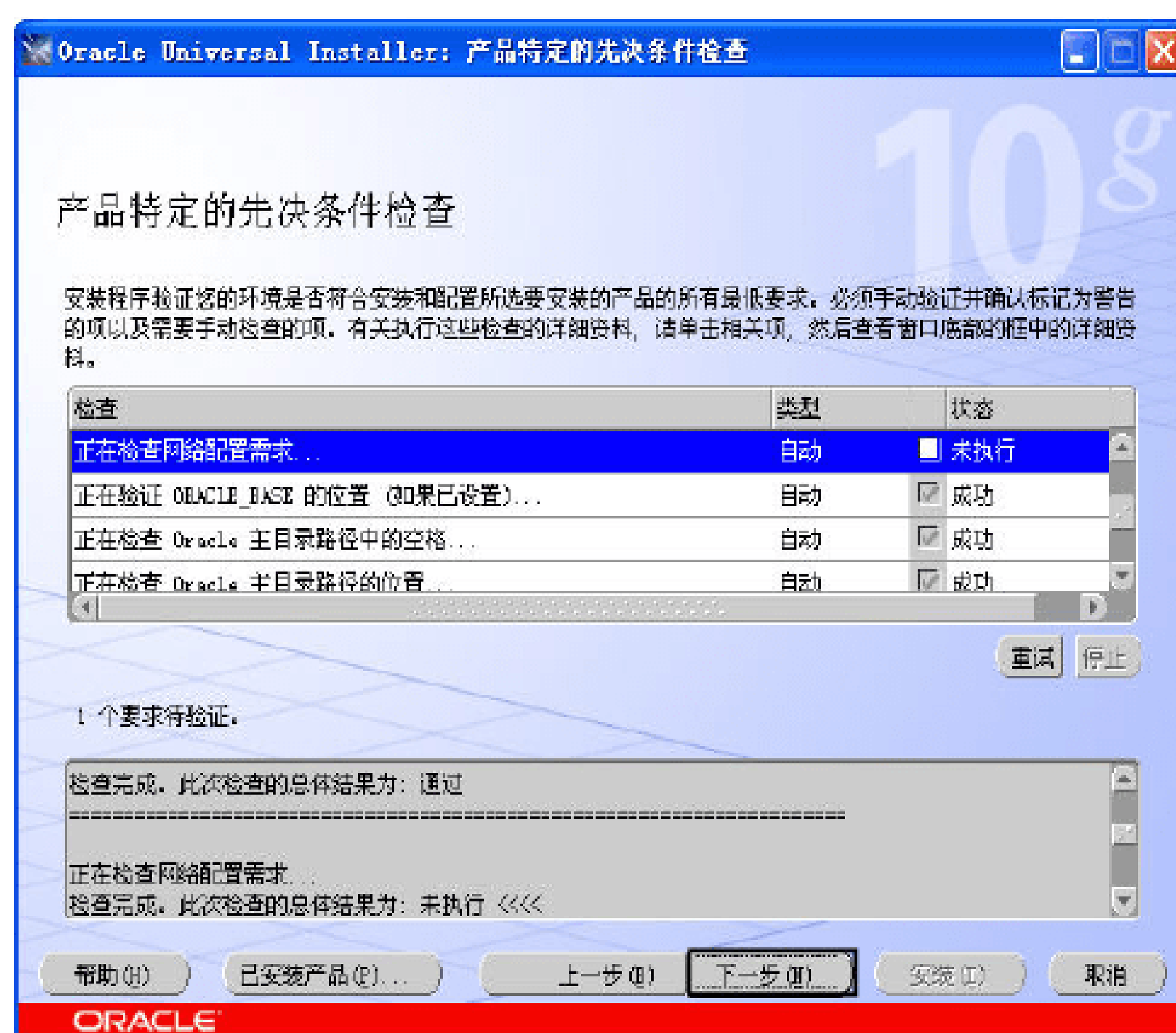


图 1.47　Oracle 文件定位

（3）单击“下一步”按钮后，弹出如图 1.48 所示的 Oracle 安装对话框。

（4）在该对话框中单击“安装”按钮，过一段时间就会出现安装结束对话框，如图 1.49 所示。单击“退出”按钮，完成 Oracle 数据库服务器的安装。

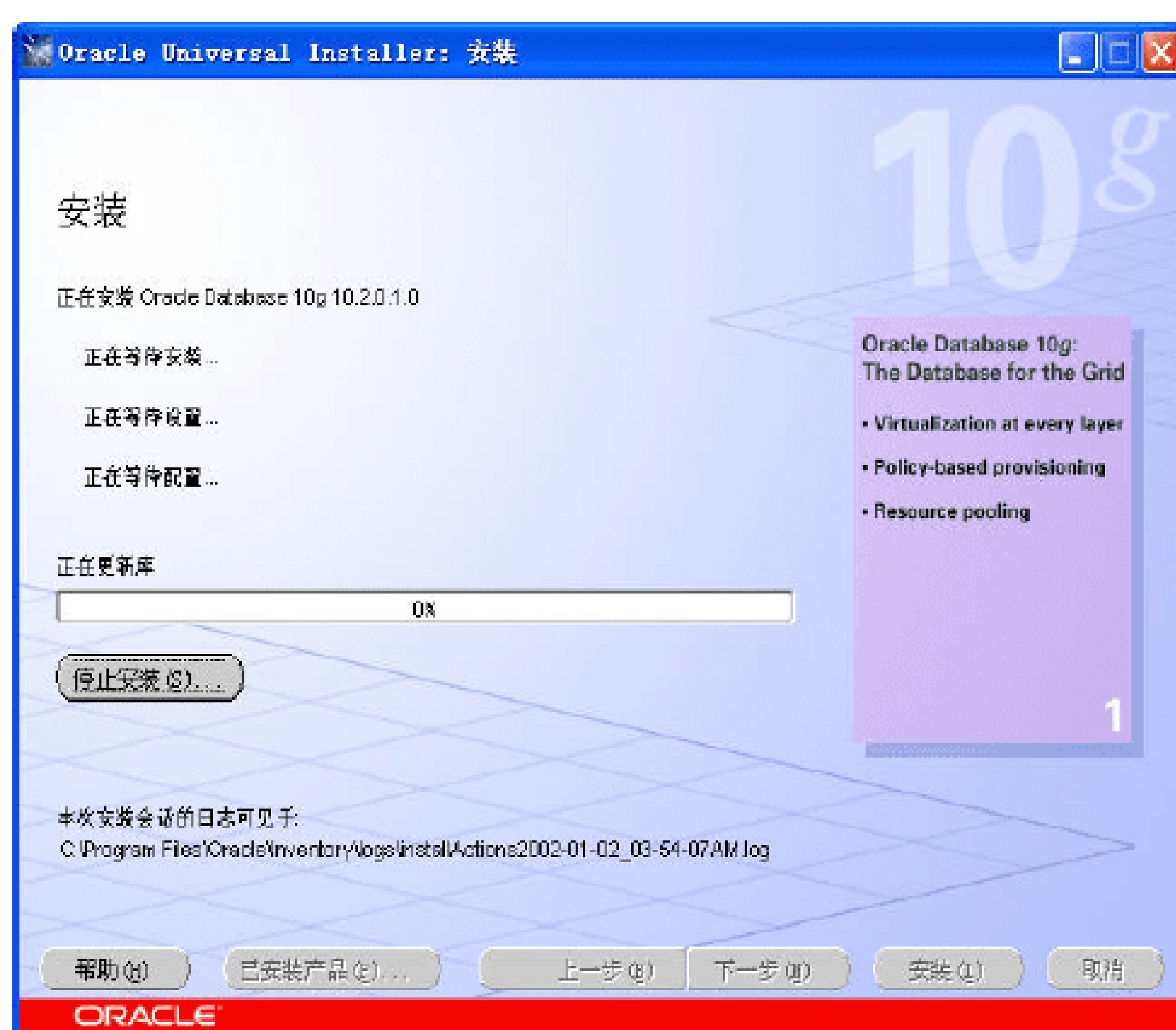


图 1.48　Oracle 安装界面

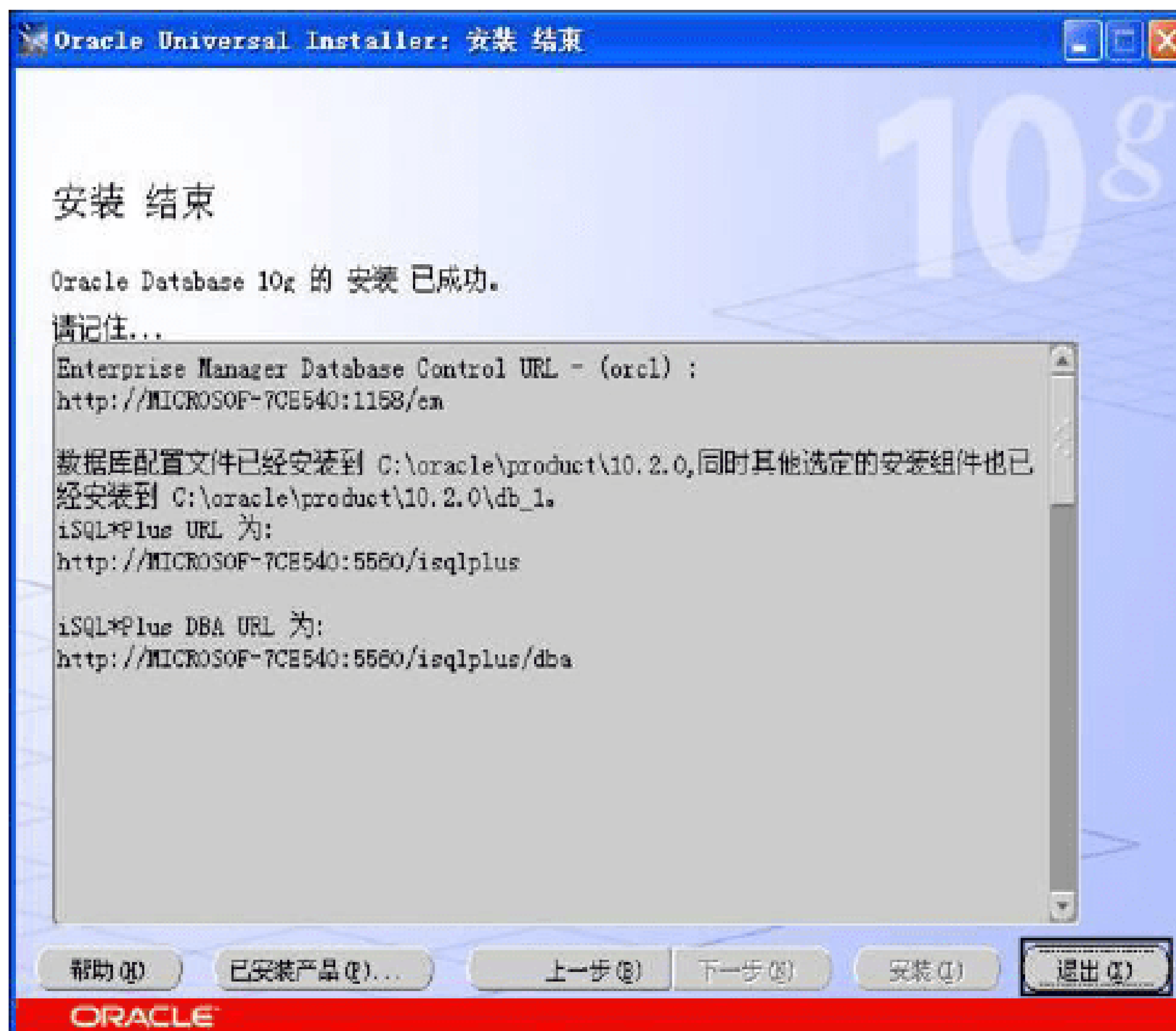


图 1.49　完成 Oracle 安装

1.2.11　安装数据库服务器 Oracle 客户端

1.2.10 节介绍了如何安装数据库 Oracle 服务器，安装完数据库 Oracle 服务器后就可以安装 Oracle 客户端以便于来操作数据库服务器。具体的安装步骤如下。

（1）双击 Oracle 客户端安装程序，接着就会使用 Windows Installer 开始安装过程，如图 1.50 所示。

（2）单击“下一步”按钮后，弹出“指定主目录详细信息”对话框，如图 1.51 所示。在该对话框中可以进行 Oracle 客户端安装文件的选择。

（3）单击“下一步”按钮后，弹出 Oracle 安装对话框。在该对话框中单击“安装”按钮，就会自动进行安装，如图 1.52 所示。过一段时间就会出现常见配置对话框。

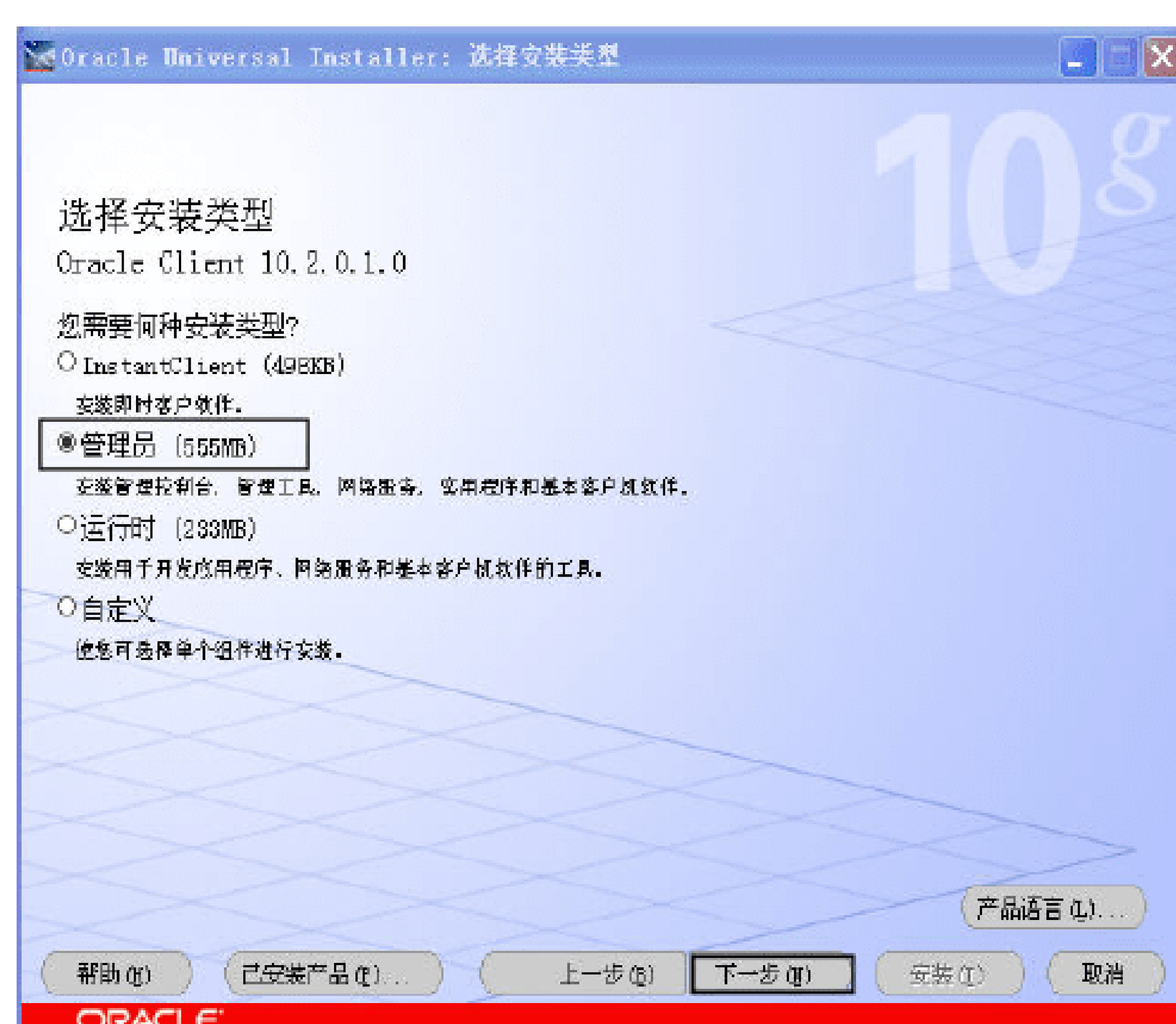


图 1.50　Oracle 选择安装类型

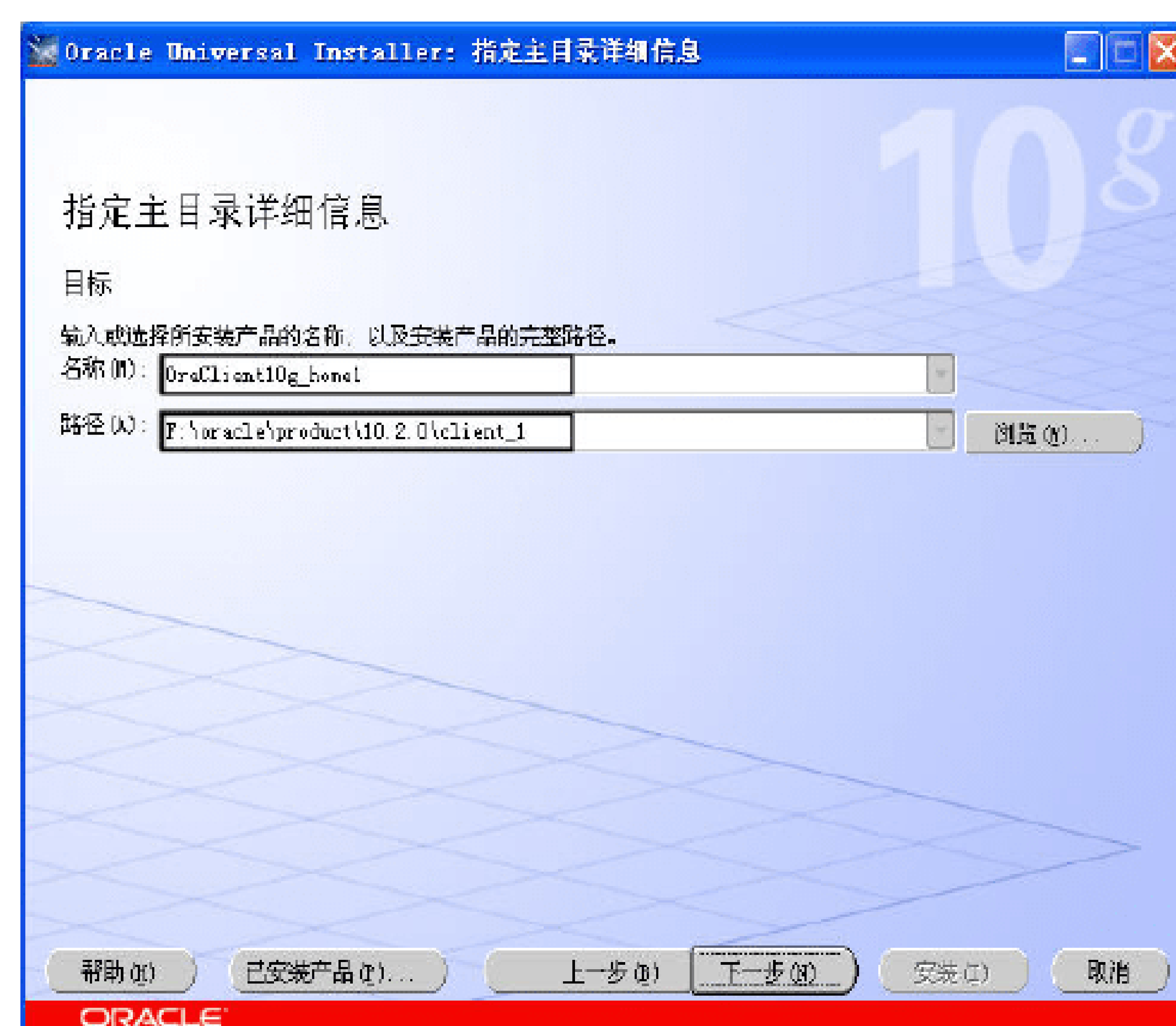


图 1.51　Oracle 客户端文件选择

（4）在如图 1.53 所示的配置对话框中，选择“执行典型配置”复选框，单击“下一步”按钮就会完成 Oracle 数据库命名方法配置。接着就会进入图 1.54 所示的“Oracle Net 配置完毕”对话框。

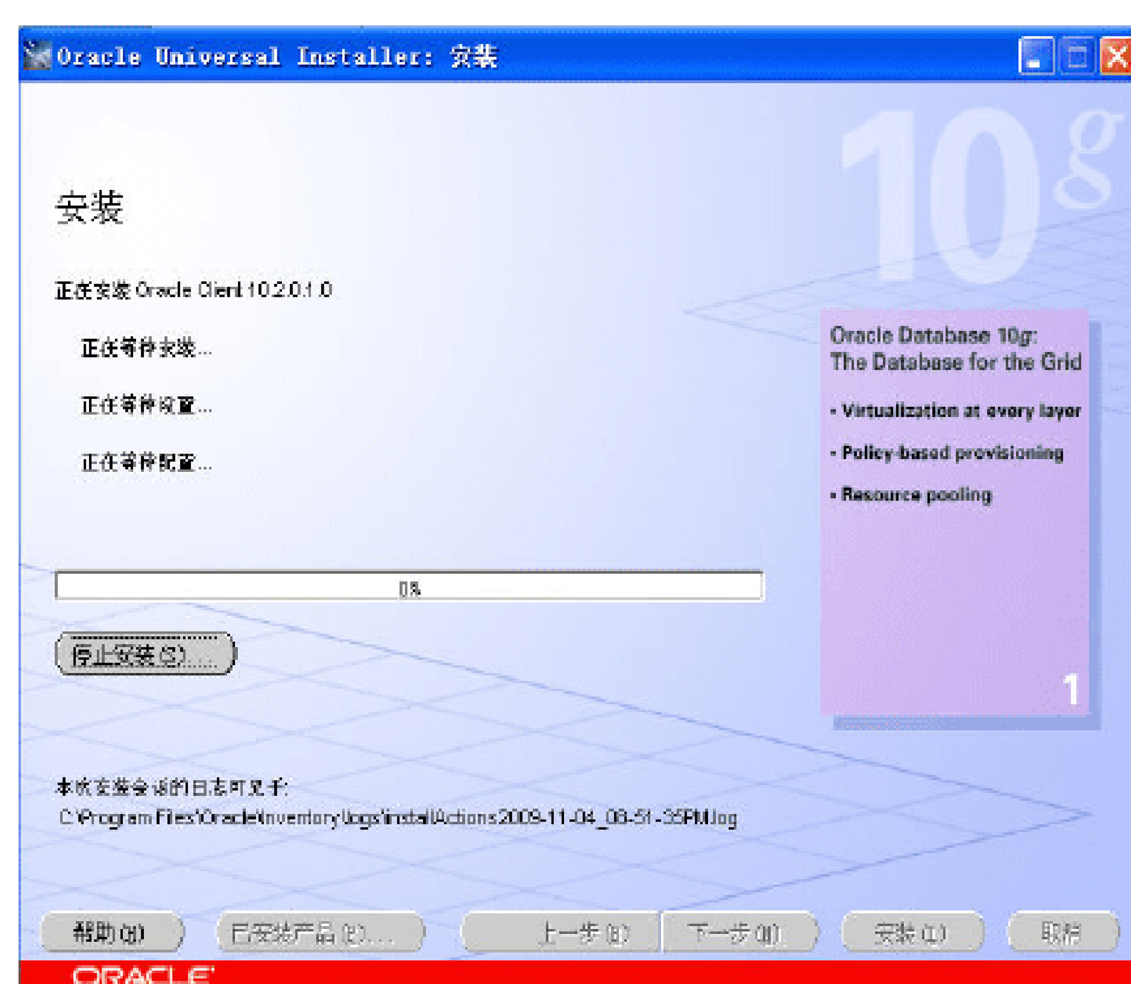


图 1.52　Oracle 客户端安装界面



图 1.53　命名方法配置

（5）单击“完成”按钮，弹出 Oracle 客户端的“安装结束”对话框，如图 1.55 所示。单击“退出”按钮结束 Oracle 客户端的安装。

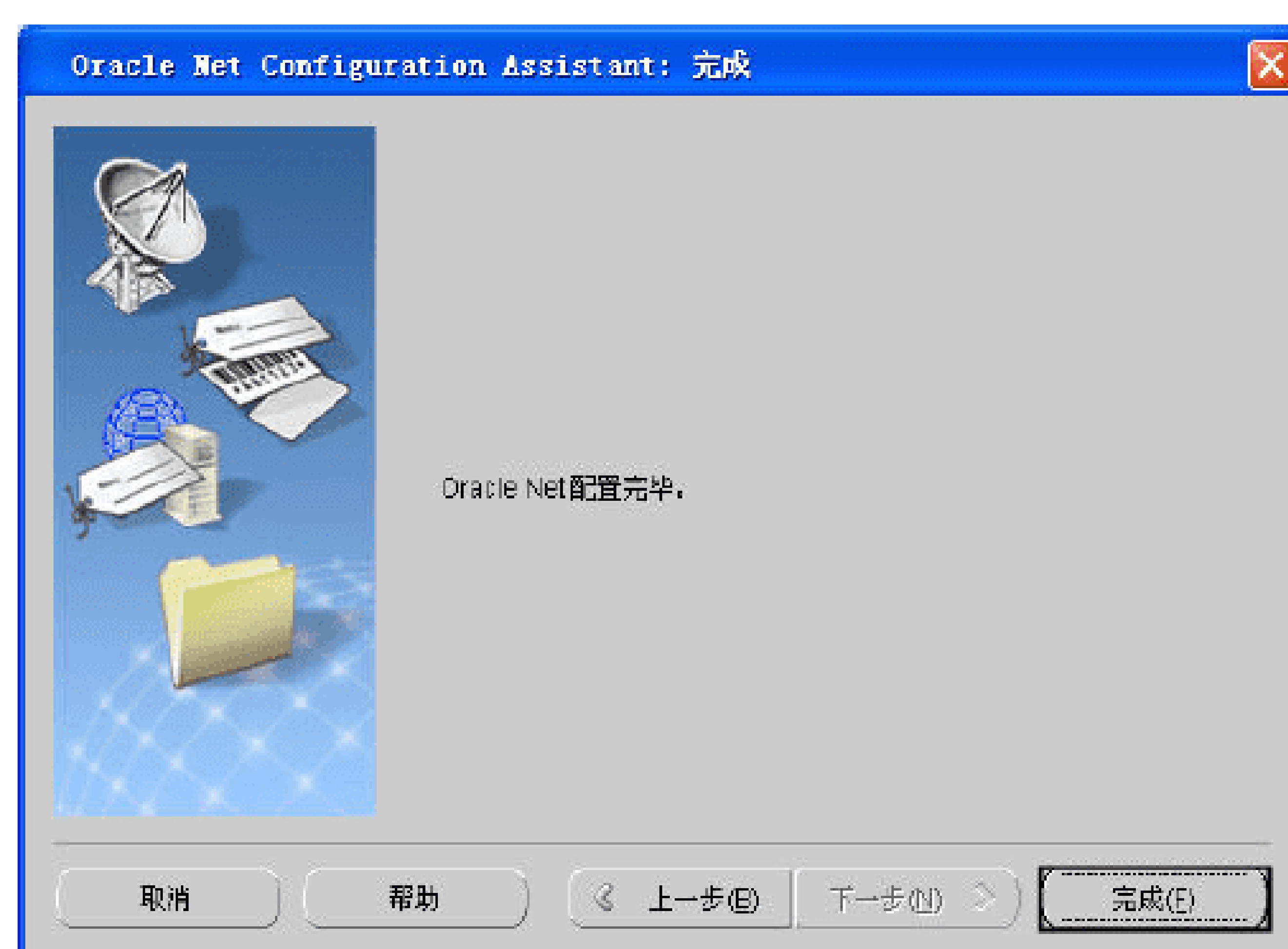


图 1.54　Oracle Net 配置完毕

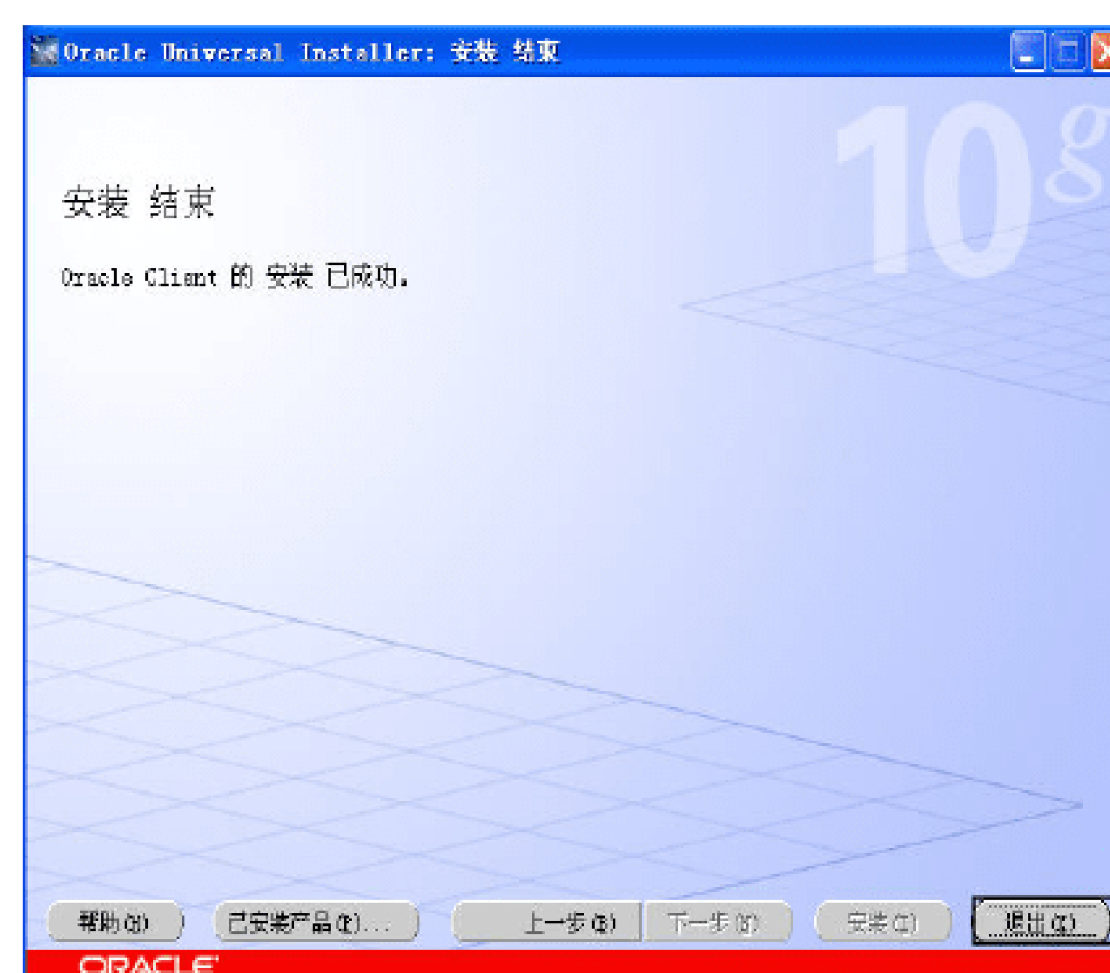


图 1.55　安装结束对话框

1.3　基础技术简单简介

不管开发 Java Web 任何方面的程序，都离不开基础编程。所谓的基础编程包括 Servlet 服务器端编程、JSP 主流网站开发技术和 JavaBean 组件技术。

1.3.1　Servlet 服务器端编程

Servlet 技术是一种用来实现动态网页的解决方案，所谓动态网页就是在不同时刻或不同条件下访问 Web 服务器上同一个页面时，浏览器会获得不同的内容。为了了解动态网页，本节将从客户端和服务器端的软件如何工作来讲解。

当浏览器访问 Web 服务器上的某个页面时，它所接收到内容就是 Web 服务器通过网络传送过来的字符流。在实际的运行时，浏览器并不关心传过来的字符流如何产生，而只是把传过来的字符流当作一个网页文档的内容来处理。所以不管任何类型的网页，浏览器都会以相同的形式处理。

当服务器接收到浏览器的一个请求时，就会从某个 HTML 文件中读取或者由一个程序动态创建字符流来响应请求。如果服务器的响应是在浏览器访问时由程序临时动态产生的，那么这个网页就是动态网页。由于网页每次都是临时产生的，所以浏览器每次显示的内容都不相同。

虽然使用客户端脚本程序或 Flash 动画，都可以在浏览器的显示效果上出现动态现象，但是这种效果是浏览器执行的结果，而不是网页的源文件内容改变后的结果，所以注意区分动态网页与这些技术的区别。

在开发软件的时候，程序员经常会在两种系统架构中进行选择，即 C/S 架构和 B/S 架构。所谓 C/S 架构（客户机/服务器架构）就是 Client/Server 的简写，其是早期出现的一种分布式架构。而 B/S 架构（浏览器/服务器架构）就是 Browser/Server 的简写，其是随着 Internet 技术的兴起对 C/S 架构的一种变化和改进的架构。

C/S 架构一般采用两层软件组件，一层是客户端程序；另一层就是服务器端的数据库，如图 1.56 所示。B/S 架构就是通过浏览器与网站系统进行交互，而网站系统却可以实现管理，如图 1.57 所示。

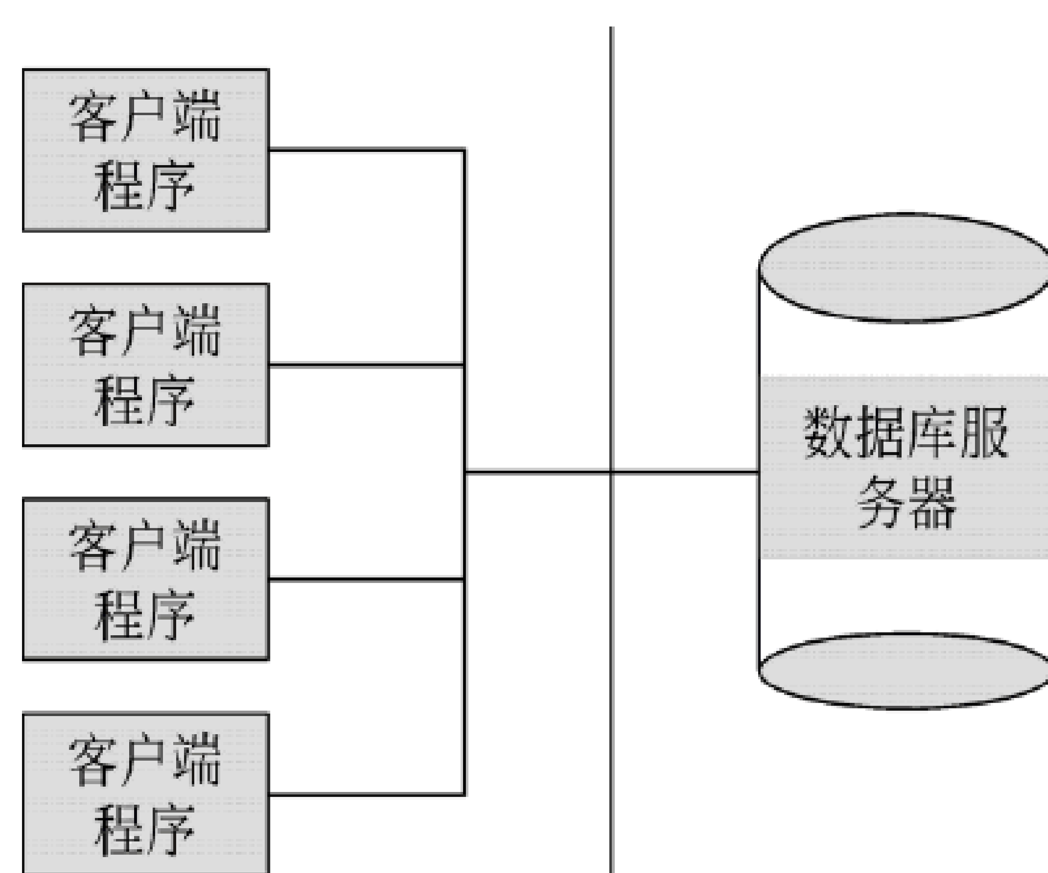


图 1.56　C/S 架构

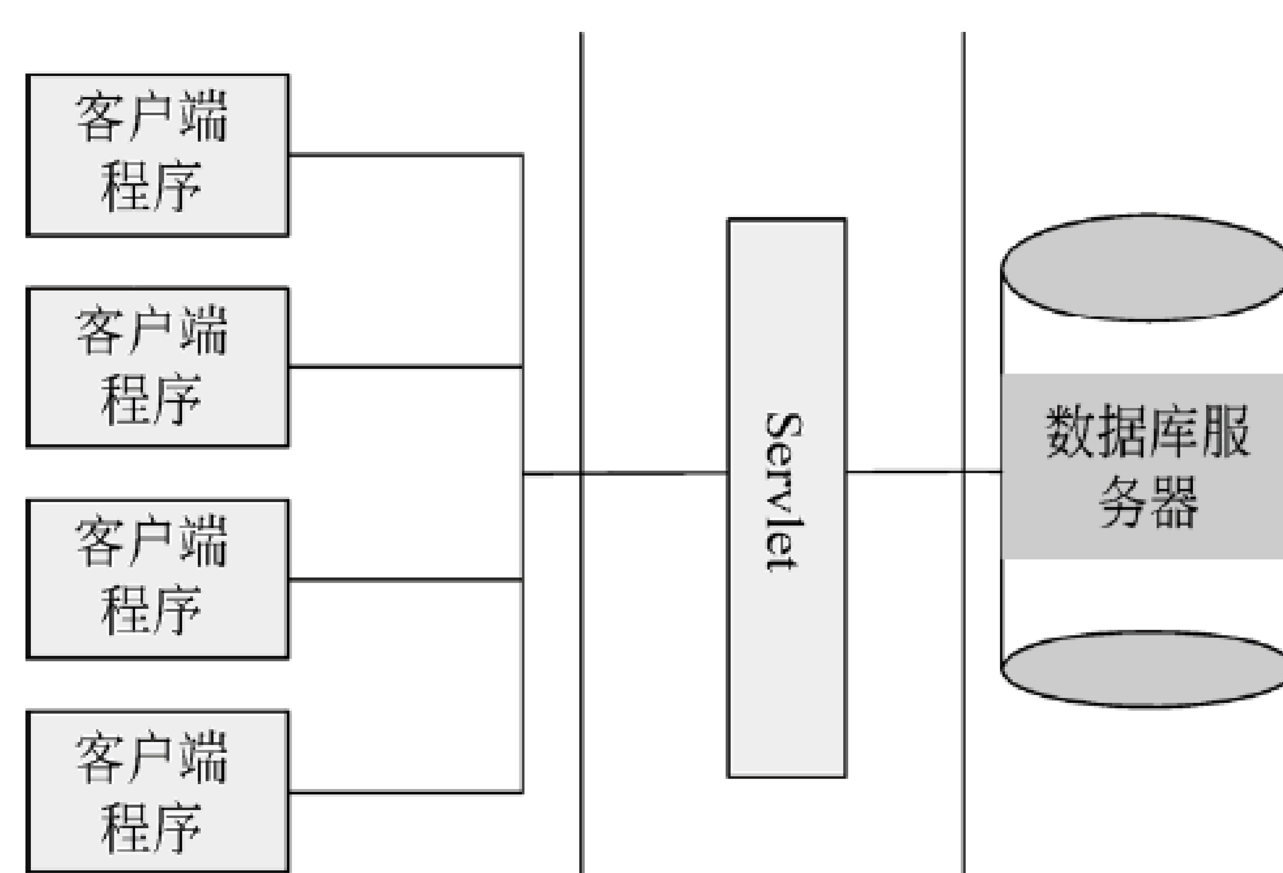


图 1.57　B/S 架构

用来处理动态网页程序的服务器程序被称为引擎，一般来说浏览器先把请求发送给引擎，然后引擎把浏览器信息传递给动态网页程序。经过处理，然后把处理结果经过引擎返回给浏览器。其具体运行过程如图 1.58 所示。

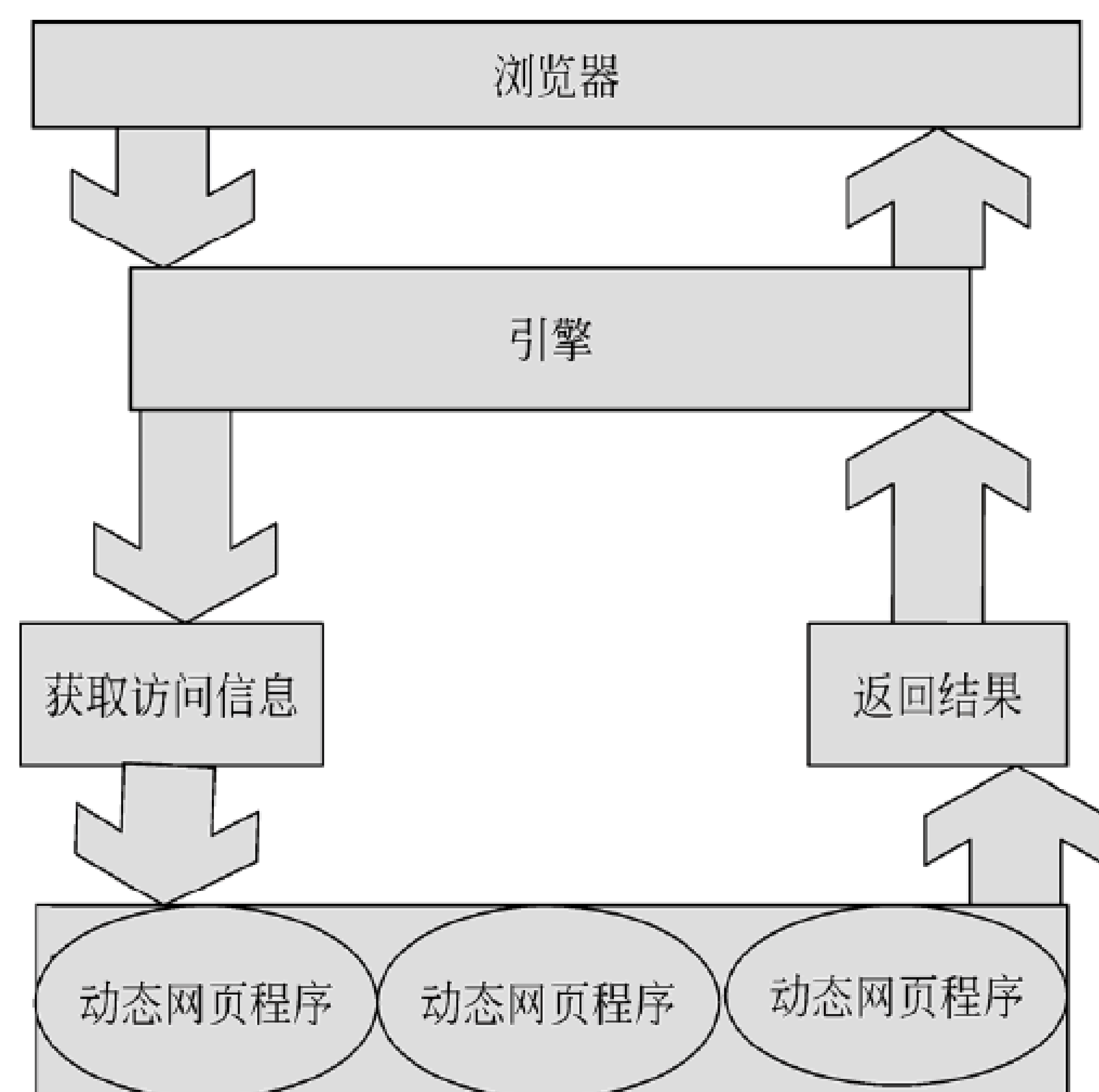


图 1.58　动态网页程序的运行过程

目前好多公司提供了动态网页解决方案，常见技术有 CGI、ISAPI、ASP、ASP.NET、PHP 和 Servlet 等。由 Sun 公司提供的 Servlet 技术存在以下优点。

- 可移植性：由于 Servlet 是用 Java 语言来开发的，因此其具有了 Java 语言的“一次编译，处处运行”的特点。即其具有良好的跨平台的表现，不论 Servlet 的操作系统是 Windows、Solaris、Linux、HP-UX 和 FreeBSD 等都能够很好地执行编写好的 Servlet 程序；
- 安全性：为了达到较高的安全性，Servlet 采用了类型检查机制、垃圾自动收集和没有指针的设计，从而避免了管理内存的问题；
- 性能：与以往的技术比较，Servlet 在性能上提高了许多。因为加载 Servlet 程序后，其对象实体是驻留在服务器的内存中；
- 功能强大：Servlet 可以开发网络、多线程、影像、分布式服务器组件、对象序列化等各种功能的程序。

1.3.2　关于 Servlet 程序的编写

Sun 公司为开发 Servlet 程序专门提供了一整套 Java 类和接口（Servlet API），使用这些接口和类可以实现 Servlet 程序开发所涉及的各种功能。在实际的运行过程中，Servlet 引擎与 Servlet 程序之间就是采用 Servlet API 进行通信，而一个 Servlet 程序就是一个在服务器端运行的 Servlet API 的 Java 类。下面用 MyEclipse 开发一个简单的 Servlet 程序，具体步骤如下。

（1）从菜单栏中选择 File | New | Web Project 命令，新建一个 Web Project 项目，在出现的对话框中进行图 1.59 所示的设置。

（2）右击刚生成的项目中的 src 目录，在弹出的快捷菜单中选择图 1.60 所示的 New|Servlet 命令，会弹出如图 1.61 所示的对话框。该对话框主要用来新建一个 Servlet 程序。

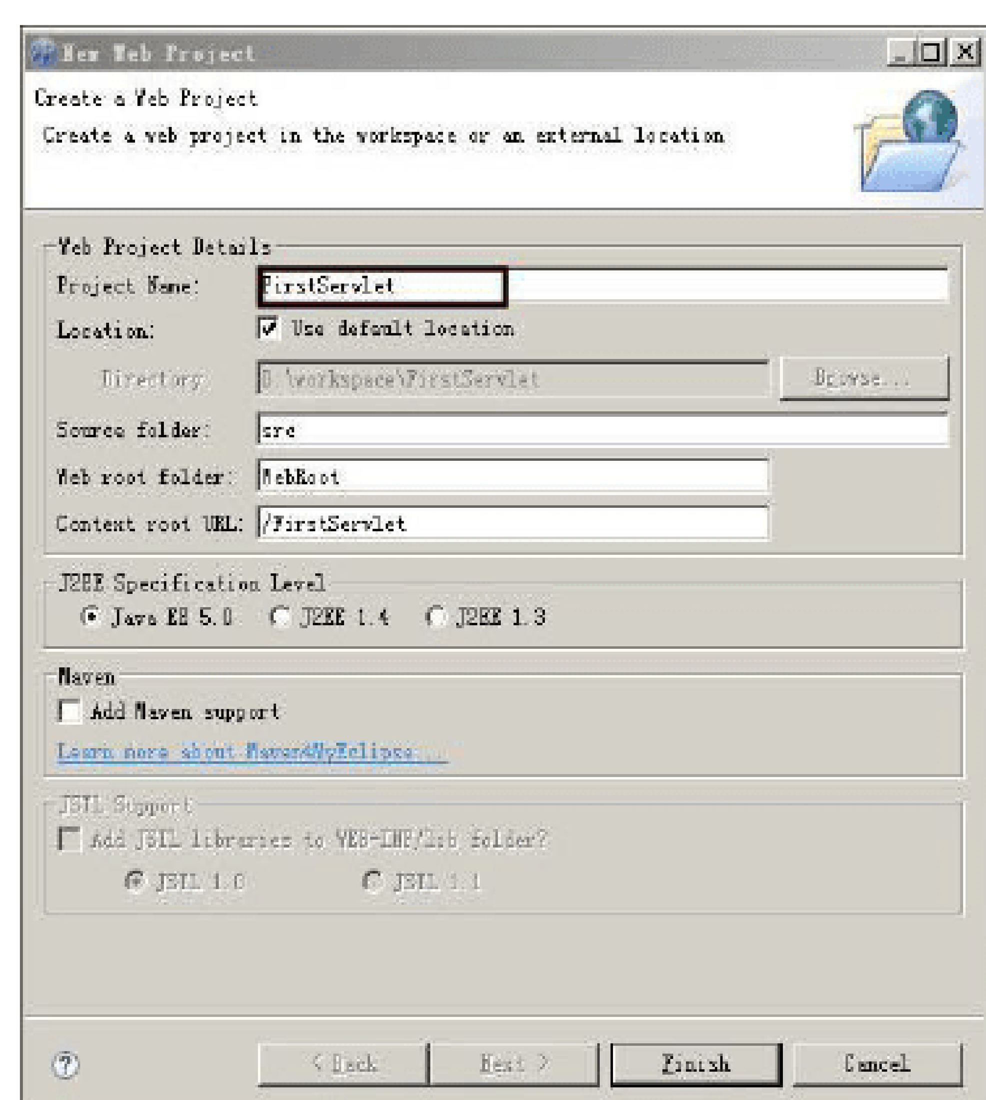


图 1.59　新建 Web Project

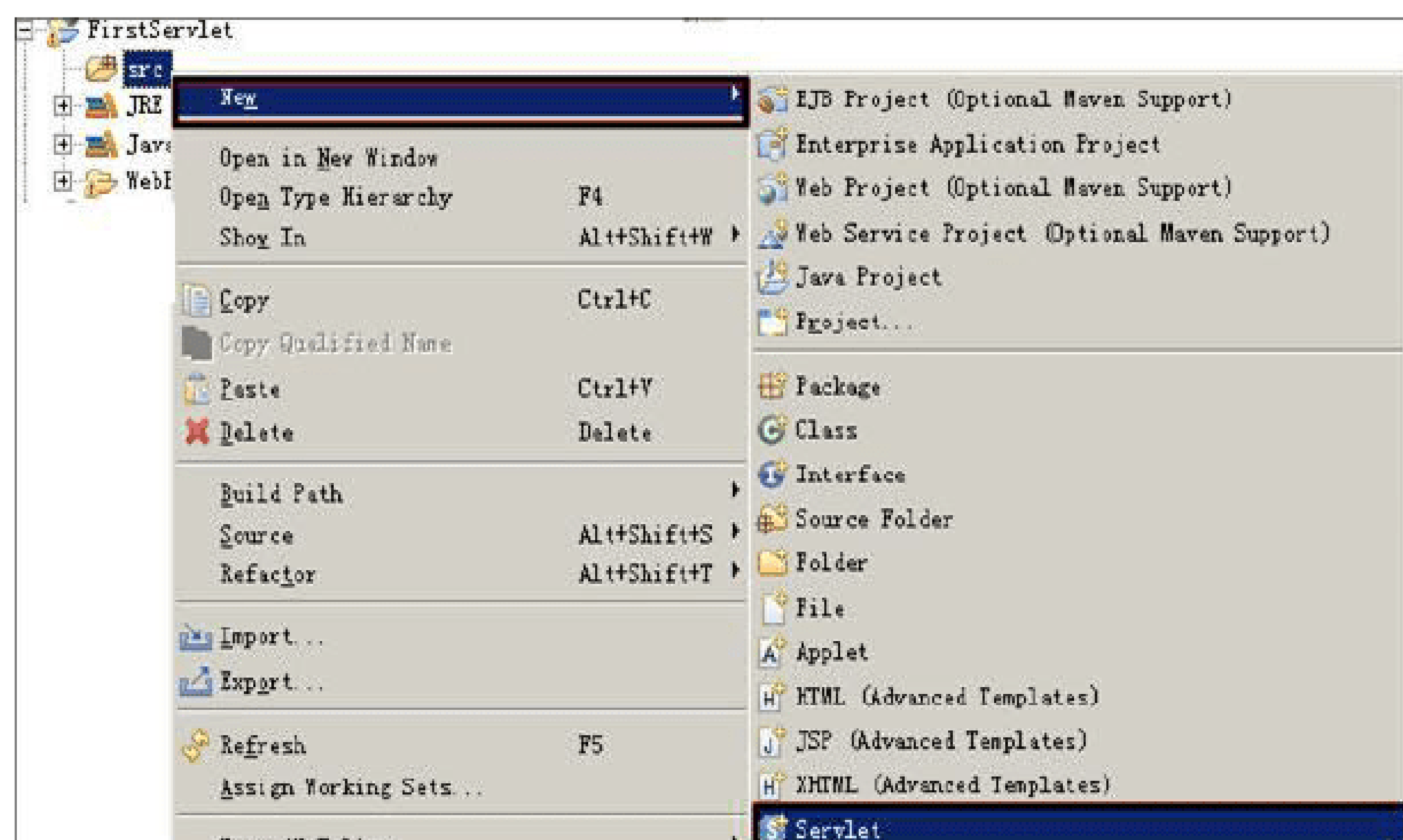


图 1.60　新建 Servlet 程序

（3）在创建 Servlet 程序的对话框中，做出如图 1.61 所示的选择后（Package 文本框中的内容用来设置 Servlet 程序文件的包），单击 Next 按钮会弹出如图 1.62 所示的对话框。在该对话框中可以对 web.xml 文件内容进行设置。

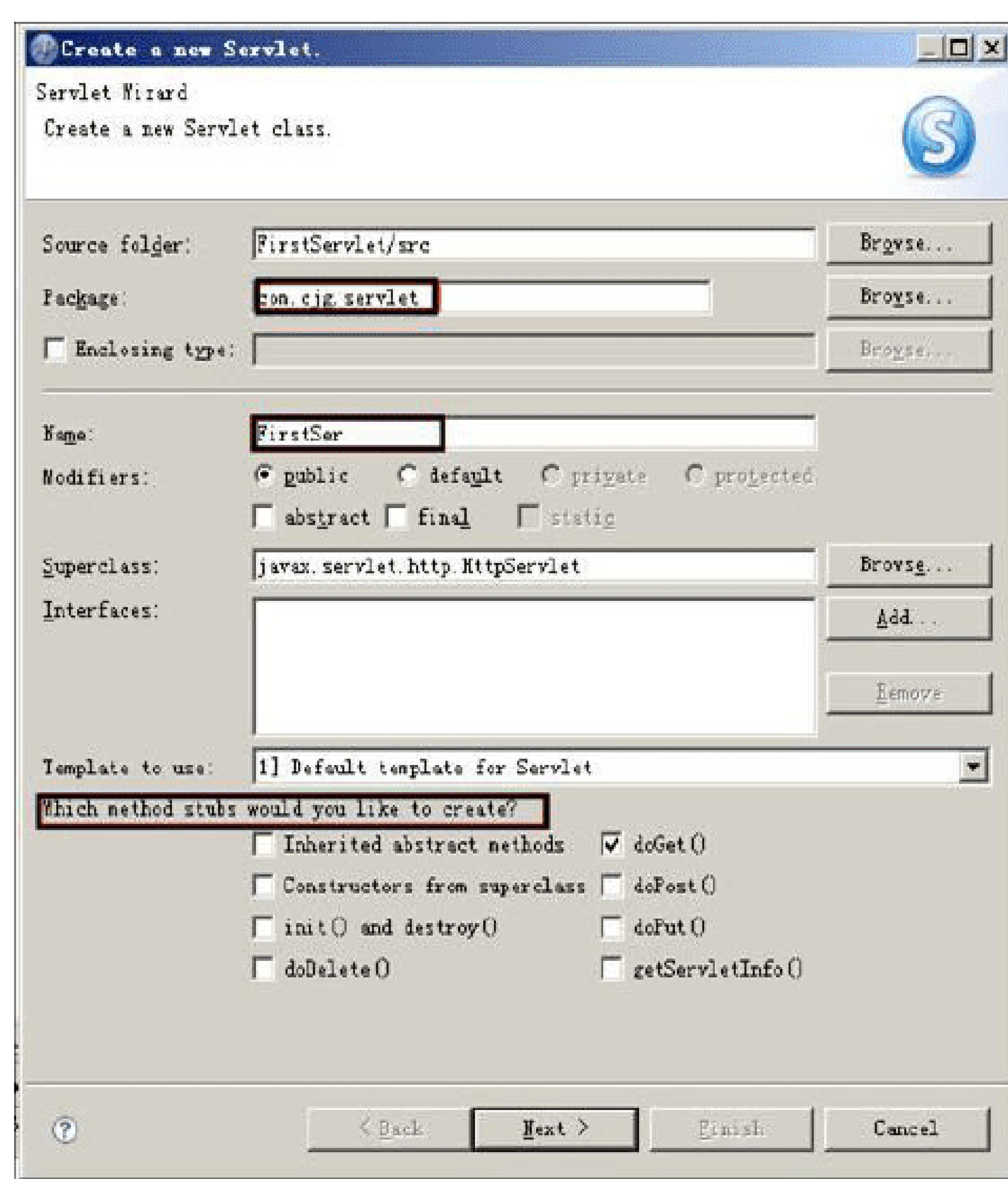


图 1.61　创建 Servlet 程序

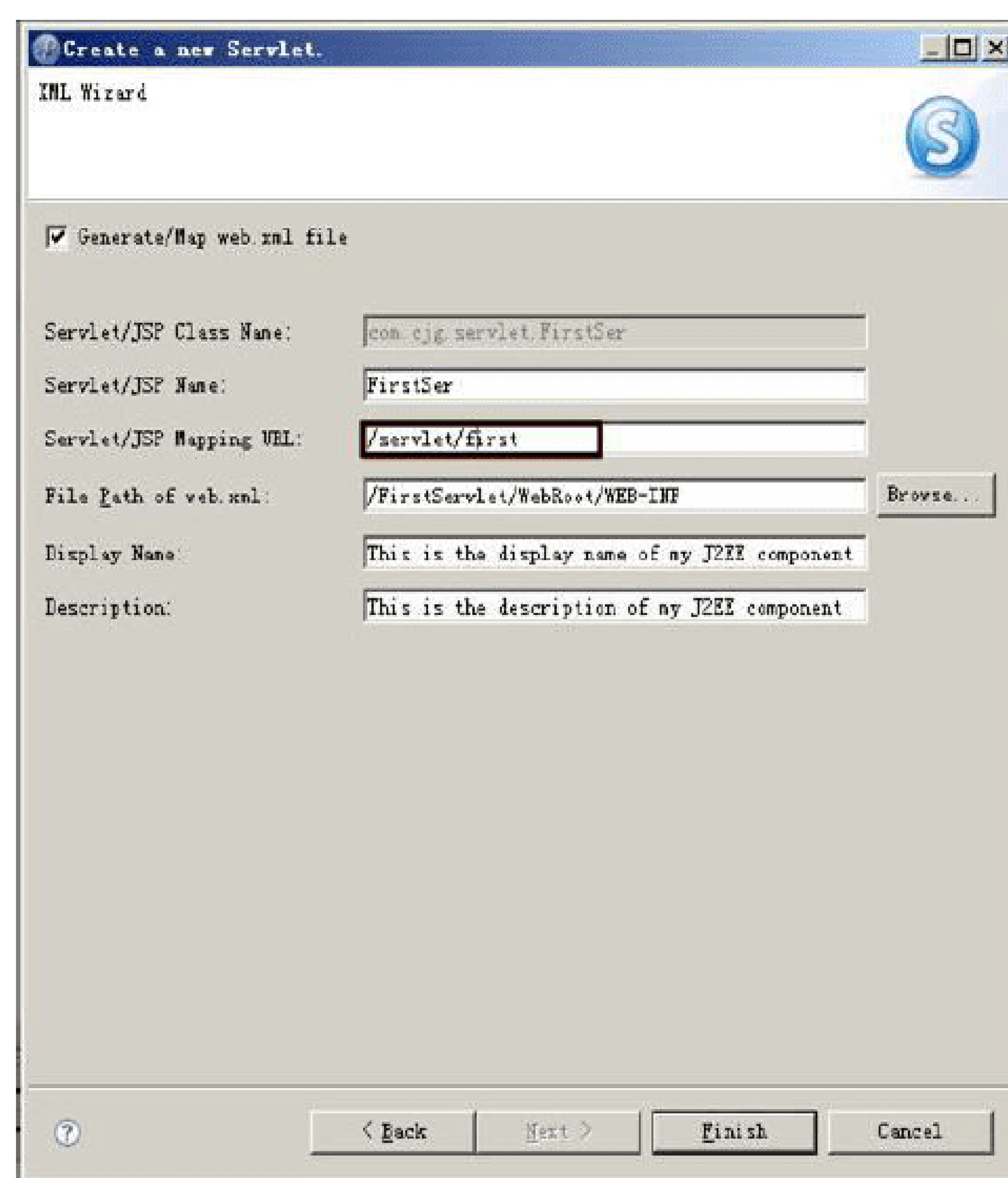


图 1.62　设置 web.xml 文件内容

（4）在设置 web.xml 文件内容的对话框中做出图 1.62 所示的修改后，单击 Finish 按钮，就完成了创建 Servlet 程序的向导。此时当前项目的目录结构如图 1.63 所示。

（5）打开文件夹 FirstServlet/src 下的 FirstSer.java 文件，代码 1.1 演示了如何实现一个简单的 Servlet 程序。

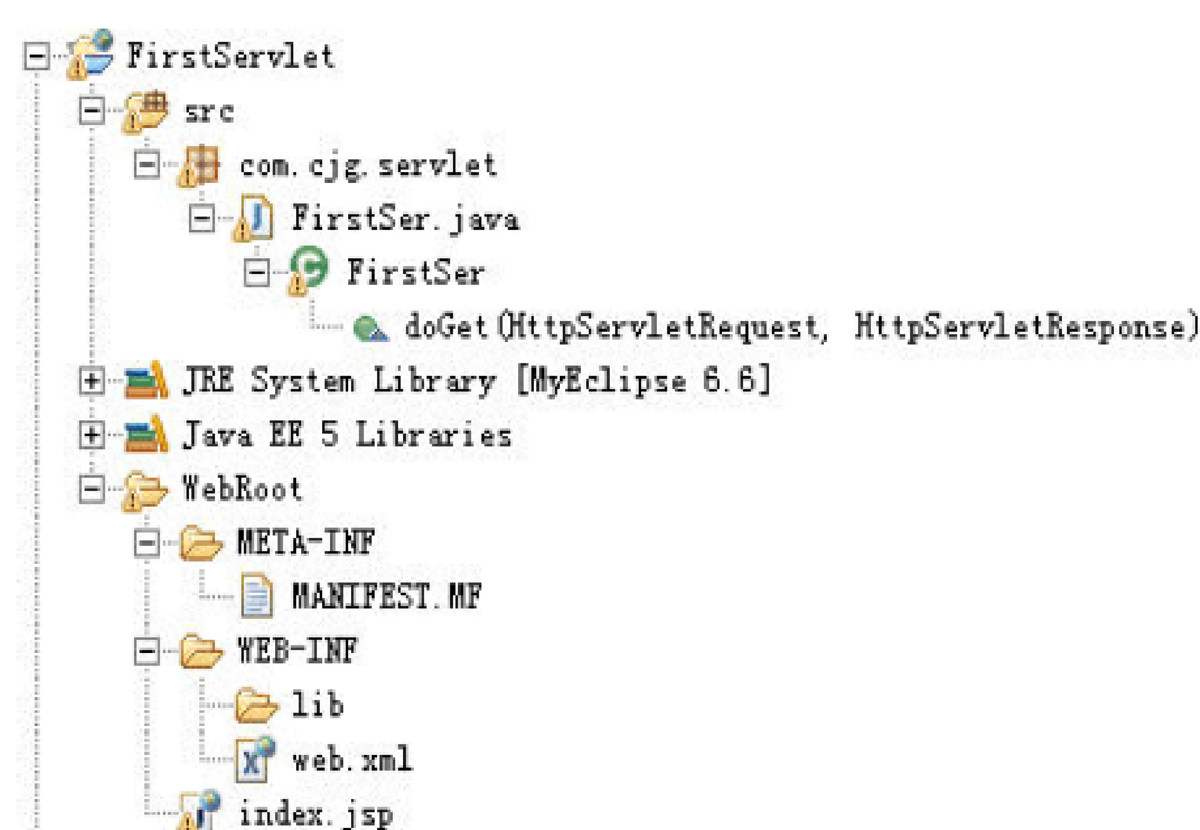


图 1.63　Servlet 程序的目录结构

代码 1.1　第一个 Servlet 程序：FirstSer.java

```
package com.cjg.servlet;
//包的引入
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
//Servlet 程序类
public class FirstSer extends HttpServlet {
    private static final long serialVersionUID = 1L;
                                                //为该 Servlet 设置一个 UID 号
    // doGet()方法的实现
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
                                                //设置输出流的类型和编码类型
        PrintWriter out = response.getWriter();//得到输出流
        out.println("<html>");
        out.println("<head><title>第一个 Servlet 程序</title></head>");
        out.println("<body>");
        out.println("第一个 Servlet 程序！");
        out.println("</body>");
        out.println("</html>");
        out.flush();                            //清除缓冲区
        out.close();                            //关闭输出流
    }
}
```

【代码解析】

- 首先必须导入包 javax.servlet.*、javax.servlet.http.*，其中第一个包中存放与 HTTP 协议无关的一般性的 Servlet 类。而第二个包除了继承 javax.servlet.*之外，还增加了与 HTTP 协议有关的功能；
- 除了引入相应的包外，编写的 Servlet 类还需要继承 java.servlet.Servlet 接口，API 文档推荐继承 javax.servlet.GenericServlet 或 javax.servlet.http.HttpServlet。如果编写

的 Servlet 类与 HTTP 协议有关，则用后者；否则相反。

打开文件夹 FirstServlet/WebRoot/WEB-INF 下的 web.xml 文件，代码 1.2 实现了对该项目的必要设置。

代码 1.2　配置 web.xml：web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app 2 5.xsd">
  <servlet>                                    <!--注册 Servlet -->
    <description>This is the description of my J2EE component</description>
    <display-name>This is the display name of my J2EE component</display-name>
    <servlet-name>FirstSer</servlet-name>      <!--Servlet 注册名-->
    <servlet-class>com.cjg.servlet.FirstSer</servlet-class>
                                               <!--Servlet 完整类名-->
  </servlet>
  <servlet-mapping>                            <!--映射 Servlet -->
    <servlet-name>FirstSer</servlet-name>      <!--Servlet 注册名-->
    <url-pattern>/servlet/first</url-pattern>  <!--Servlet 的对外访问路径-->
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

【代码解析】

- Servlet 程序必须在 Web 应用程序的 web.xml 文件中进行注册和映射访问路径，才能被外界访问。
- 元素<servlet>用来注册 Servlet 程序，其主要有两个子元素：<servlet-name>和<servlet-class>。元素<servlet-name>用来设置 Servlet 的注册名称，该名称可以是一个任意合法的名称，但是绝对不能与其他 Servlet 注册名称相冲突。元素<servlet-class>用来指定当前注册的 Servlet 程序的完整类名，注意完整类名中要包括该类的包名。
- 元素<servlet-mapping>用来映射一个已注册的 Servlet 的一个对外访问路径，其主要目的是因为用户只有通过 Servlet 所映射的对外访问路径，才能访问该 Servlet，而不是使用 Servlet 名称来访问。该元素主要有两个子元素：<servlet-name>和<url-pattern>。元素<servlet-name>用来指定已经注册过的 Servlet 名称，元素<url-pattern>用来设置 Servlet 的对外访问路径。

注意：对外访问路径必须以“/”符号开头，表示当前 Web 应用程序的根目录。

（6）单击工具栏上的按钮，弹出如图 1.64 所示的对话框以实现把该项目发布到服务器。然后单击工具栏上的按钮，在弹出的选项中做出如图 1.65 所示的选择，以实现启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/FirstServlet/

servlet/ first，该地址解析如图 1.66 所示。这时运行结果如图 1.67 所示。

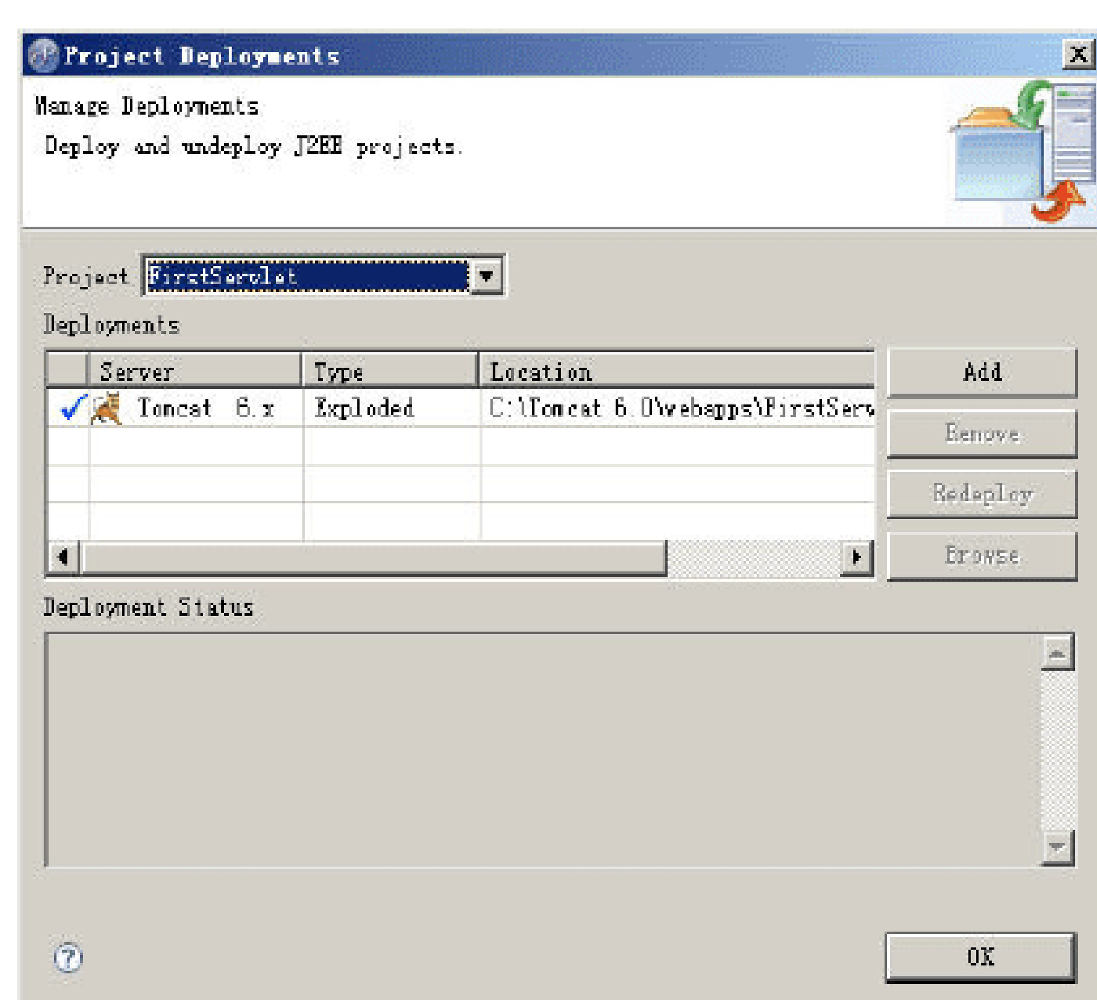


图 1.64　发布应用程序图

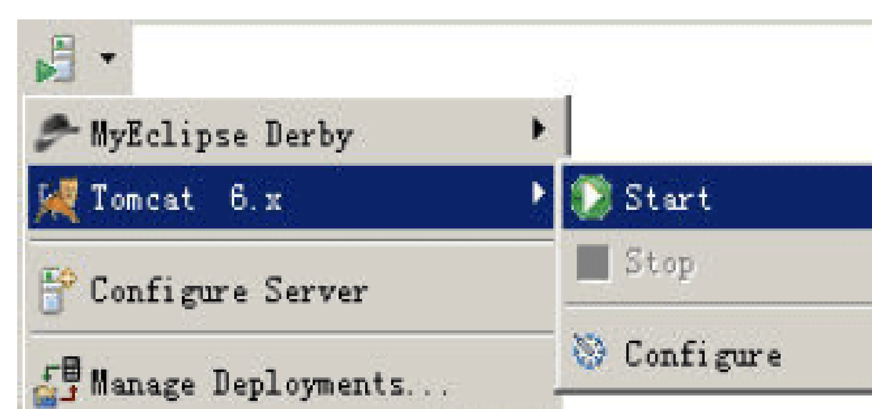


图 1.65　启动服务器



图 1.66　地址解析



图 1.67　运行结果

1.3.3　JSP 主流网站开发技术

JSP 文件的后缀名必须为 jsp，其内容包括 HTML 语句和 Java 代码。Java 代码一般被嵌套在<%和%>中，称为脚本片段，而没有嵌套在<%和%>中的内容称为模板元素。脚本片段被当做 Java 程序执行，而模板元素则被输出给浏览器执行。下面用 MyEclipse 开发一个简单的 JSP 程序，具体步骤如下。

（1）从菜单栏中选择 File|New|Web Project 命令，新建一个 Web Project 项目，在弹出的对话框中进行如图 1.68 所示的设置。

（2）右击项目的名称出现快捷菜单，然后做出如图 1.69 所示的选择来新建一个 JSP 程序，这时就会弹出如图 1.70 所示的对话框。Template to use 是 JSP 模板，使用它可以加快开发的速度。该对话框一般只需要修改 File Name 文本框内容。

（3）在创建 JSP 对话框中做出如图 1.70 所示的修改后，单击 Finish 按钮就会完成创建 JSP 程序的向导。这时目录结构如图 1.71 所示。

（4）打开文件夹 FirstJsp/WebRoot 下的 FirstJsp.jsp 文件，代码 1.3 演示了如何获取当前时间。

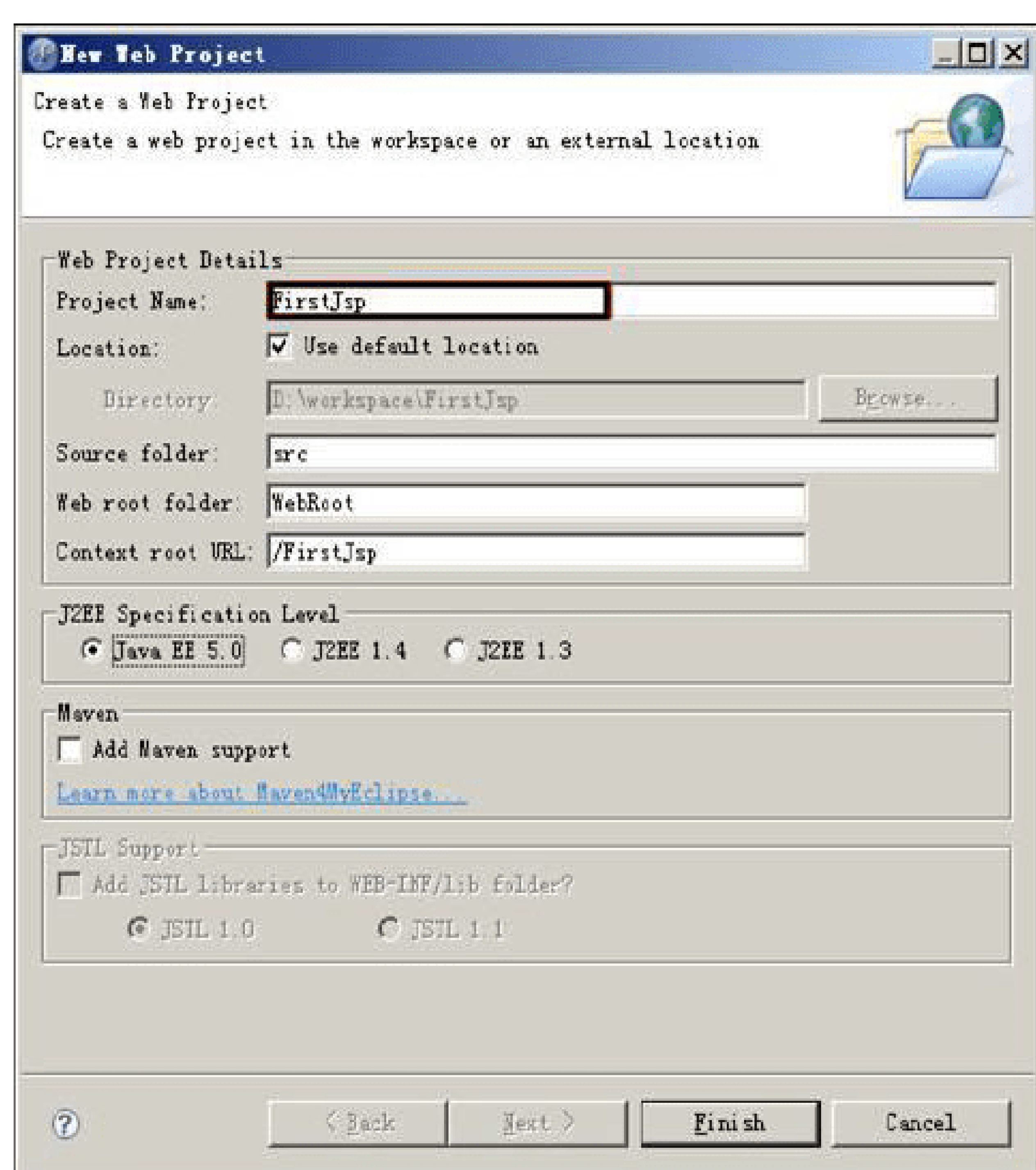


图 1.68　新建 Web Project

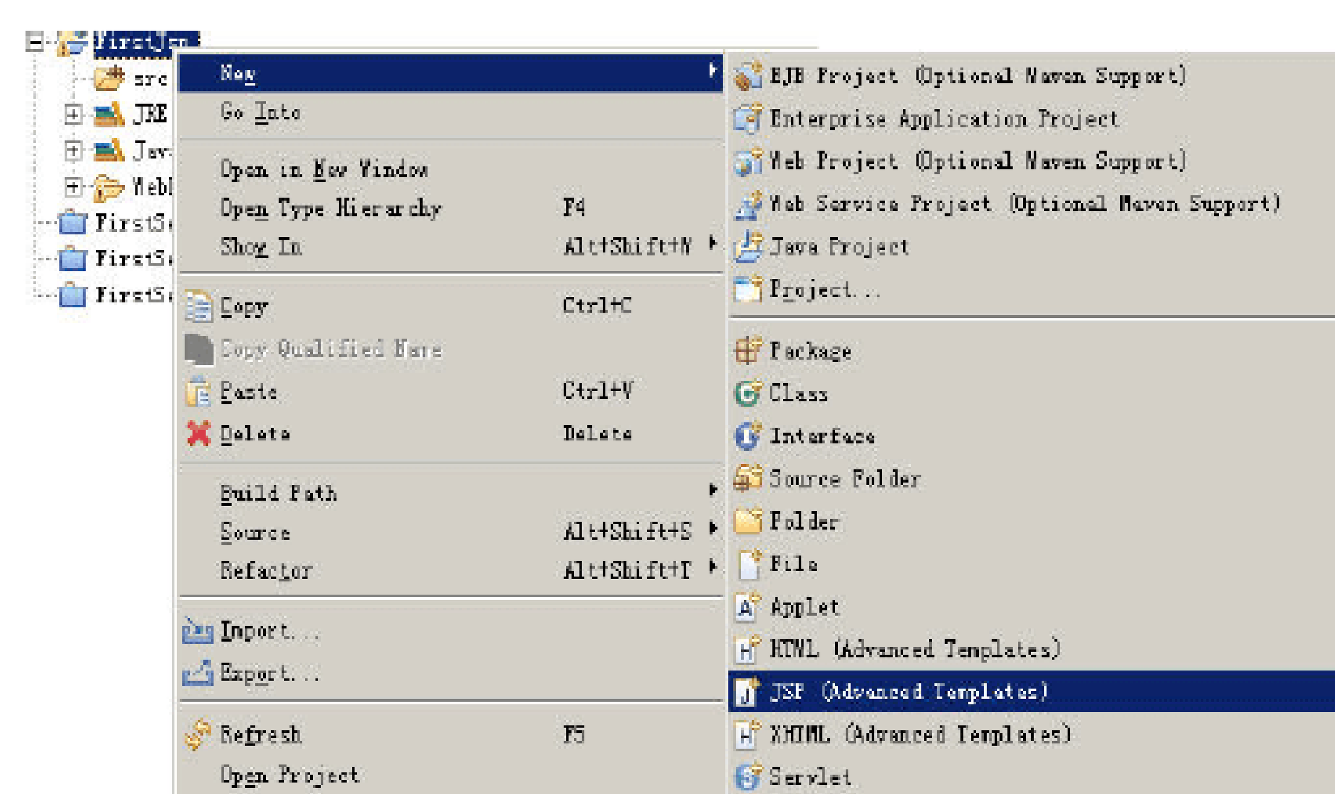


图 1.69　新建 JSP 程序

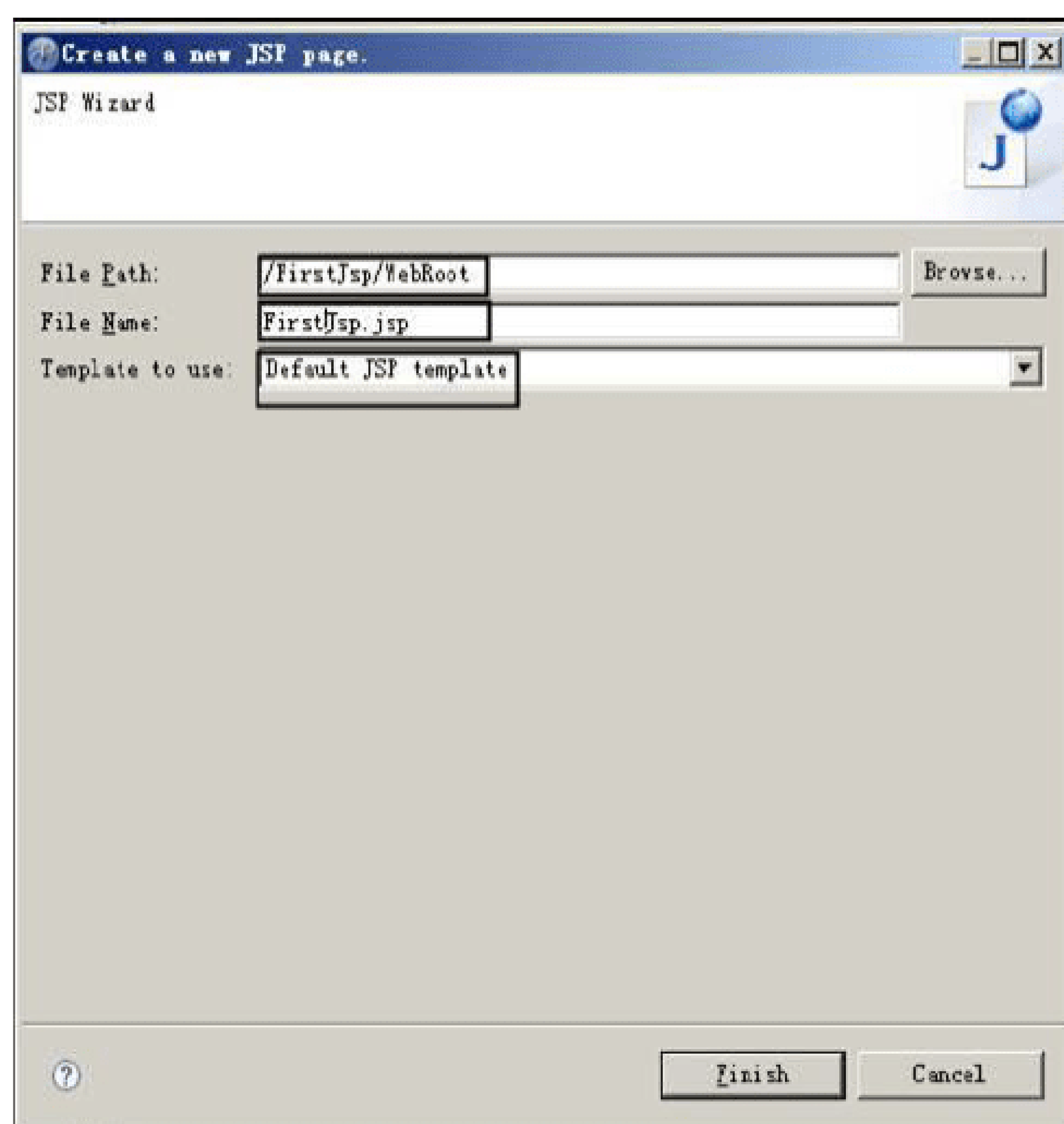


图 1.70　创建 JSP 程序

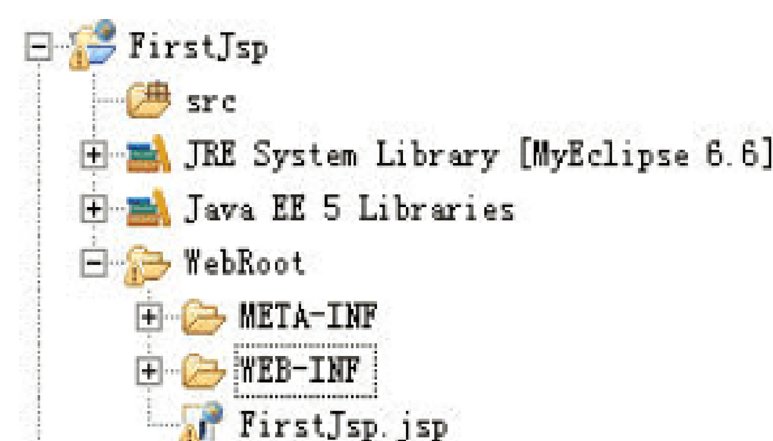


图 1.71　JSP 程序的目录结构

代码 1.3　第一个 JSP 程序：FirstJsp.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<!--HTML 部分-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>My JSP 'FirstJsp.jsp' starting page</title>
  </head>
  <body>
    现在时间是：
    <!--JSP 中的脚本片段-->
    <%
    String currTime= new java.util.Date().toString();
```

```
                                                //获取当前时间并赋值给 currTime 变量
out.println(currTime);                          //输出 currTime 变量的值
    %>
  </body>
</html>
```

（5）单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080//FirstJsp/FirstJsp.jsp，运行结果如图 1.72 所示。



图 1.72　运行结果

Web 容器接收到以 jsp 为扩展名的 URL 的访问请求时，会把该访问请求交给 JSP 引擎去处理，JSP 引擎负责解释执行 JSP 页面。当一个 JSP 页面第一次被访问时，JSP 引擎会将该页面翻译成一个 Servlet 源程序，接着再把这个 Servlet 源程序编译成 Servlet 的 class 类文件，最后由 Web 容器像调用普通 Servlet 程序一样的方式来转载和解释执行这个 class 类文件。

1.3.4　JSP 的一些基本语法

JSP 同所有其他语言类似，也有自身语法例如 JSP 的代码注释、指令（Directive）标记、声明（Declaration）标记、脚本（Scriptlet）标记、表达式（Expression）标记和动作。本节将介绍注释。JSP 语言有 4 种注释。

- “//”：对单行 Java 代码进行注释；
- “/**/”：对多行 Java 代码进行注释；
- “<%--　--%>”：对多行标记代码进行注释；
- “<! --　-->”：对多行 HTML 代码进行注释。

JSP 引擎执行 JSP 页面时，将忽略 JSP 注释中的所有内容。

在 JSP 中还存在一种叫做指令的标记，该指令标记针对的对象是 JSP 引擎，它们并不会直接产生任何可见输出，只是告诉引擎如何处理 JSP 页面中的其余部分。JSP 语言有 3 种指令标记。

- page 指令：对当前 JSP 页面的特性进行说明；
- include 指令：包含另外的 JSP 页面和 HTML 页面；
- taglib 指令：用于 JSP 技术的标记库。

指令标记的语法格式为：

```
<%@ 指令标记 属性名=“值”%>
```

下面通过一个具体的实例来讲解指令标记的作用。

（1）首先新建一个名为 Include 的 JSP 页面，代码 1.4 实现输出一些内容功能。

代码 1.4　使用指令标记：Include.jsp

```
<!-- page 指令使用-->
<%@ page language="java"%>
<%@ page pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Include 页面</title>
  </head>
  <body>
    <%out.println("include 指令的使用：我是被包含的页面内容"); %>
                                                          <!--输出一些内容-->
  </body>
</html>
```

【代码解析】

当一个 JSP 页面中设置同一条指令标记的多个属性时，可以使用多条指令标记单独设置，也可以使用同一条指令标记语句设置该指令的多个属性。所以两条 Page 指令标记语句可以改写成如下：

```
<%@ page language="java"% pageEncoding="UTF-8>
```

指令标记 Page 用来定义 JSP 页面的各种属性，该指令标记可以出现在 JSP 页面的任何地方，但是其定义的属性都是整个 JSP 页面。下面将介绍该指令标记的常见属性。

- language 属性：用来指定 JSP 页面所使用的脚本语言；
- extends 属性：用来指定 JSP 页面编译成 class 文件时所继承的父类；
- conterntType 属性：用来设置响应正文的 MIME 类型；
- pageEncoding 属性：用来设置 JSP 页面的字符所使用的字符集编码。

（2）接着再新建一个名为 DirectiveJsp 的 JSP 页面，代码 1.5 用来把 Include.jsp 页面中内容合并到当前页面。

代码 1.5　使用指令标记：DirectiveJsp.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>指令标记</title>
  </head>
  <body>
    <%@ include file="/Include.jsp" %>        <!-- include 指令使用-->
  </body>
</html>
```

【代码解析】

为了把 Include.jsp 页面中的内容合并到当前的页面中，这里使用了 include 指令标记。使用该指令标记引入外部文件的方式，叫做静态引入。该指令标记的语法格式如下：

```
<%@ include file="relativeURL" %>
```

属性 file 用于指定被引入文件的相对路径。如果以“/”开头，表示相对于当前 Web 应用程序的根目录。使用浏览器访问该页面，结果如图 1.73 所示。



图 1.73　运行结果

1.3.5　JavaBean 组件技术

所谓非可视化 JavaBean，就是没有用户界面或用户接口的 JavaBean，其只是单纯地处理一些事务，如数据运算和数据处理等，可以很好地实现业务逻辑和前台程序的分离。

可以这样理解非可视化 JavaBean，即把其当做一个黑盒子，只知道其所具备的功能，但是不知道其内部是如何运行的。这种情况跟现实生活中的自动售饮料机一样，只要投入一定的钱，就会掉下来相应的饮料，但是却不知道其内部的运行机制。通常一个标准的 JavaBean 具有如下特征：

- JavaBean 是一个公开的类；
- JavaBean 类有一个无传入参数的构造函数；
- 如果想取得或设定属性时，必须使用 getXXX()方法或 setXXX()方法。

创建一个 JavaBean 很简单，跟创建一个 Java 类很相似。关键要注意一点就是 JavaBean 中经常用 get()和 set()这样的成员方法来处理属性。代码 1.6 通过一个简单的例子演示了 JavaBean 是如何实现的。

代码 1.6　创建 JavaBean：SimpleJavaBean.java

```
public class SimpleJavaBean {                    //定义了一个公开的类
    private String firstProperty;                //定义了一个私有属性
    public SimpleJavaBean() {                    //定义了一个无参数构造函数
    }
    public String getFirstProperty() {           //为属性定义了一个 getXXX()方法
    return firstProperty;
    }
    public void setFirstProperty(String value){//为属性定义了一个 setXXX()方法
    firstProperty = value;
    }
    public static void main(String[] args) { //输出内容
        System.out.println("第一个非可视化 JavaBean!");
    }
}
```

【代码解析】

对于 SimpleJavaBean 类，其具备了 JavaBean 的 3 大特征：即由于其是一个公开的类，这具备了标准 JavaBean 的第 1 个特征；由于其有一个无传入参数的构造函数，这具备了标准 JavaBean 的第 2 个特征；由于其声明了一个 String 类型的属性：FirstProperty，为其定

义了两个方法：setXXX()和 getXXX()，这具备了标准 JavaBean 的第 3 个特征。本例的运行结果如图 1.74 所示。

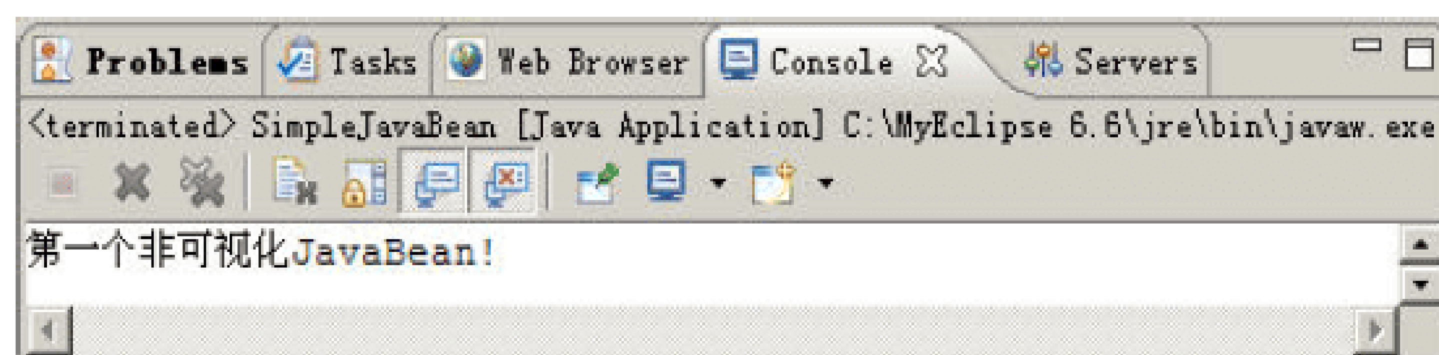


图 1.74　标准 JavaBean 的运行结果

1.3.6　JavaBean 的属性——简单属性

JavaBean 的属性与 Java 类中的属性概念有本质的不同，读者要注意区别。从 SimpleJavaBean.java 文件就可以看出 JavaBean 的属性是以方法定义的形式出现的，而对属性赋值是通过 setXXX()方法（属性修改器），对属性值的读取是通过 getXXX()方法（属性访问器）。

属性修改器和属性访问器的命名是有具体规定的，即必须以小写的 set 和 get 前缀开始，后跟属性名，并且属性名的第一个字母要大写。例如，firstProperty 属性的修改器名称为 setFirstProperty，属性访问器的名称为 getFirstProperty。

JavaBean 中的属性为什么会有如此多的限制呢？这是因为 JavaBean 的属性名是根据 getXXX()方法与 setXXX()方法来生成的，这些方法前缀 get 和 set 后的部分即为属性名。

如果一个属性既有属性访问器又有属性修改器，则该属性为读写属性。读写属性是最常见的属性。有些属性只有属性访问器，称为只读属性。例如，人的性别是不能修改的，只能根据实际情况读取。但是有些属性只有属性修改器，称为只写属性，这种属性比较少见。

根据属性的复杂程序，JavaBean 分为简单属性和复杂属性。代码 1.7 是一个代表用户信息的 JavaBean，在该 JavaBean 中设计了两个简单的属性：name（名称）和 old（年龄）。

代码 1.7　简单属性：People.java

```
public class People {
    //定义了两个属性 name 和 old
    private String name;
    private int old;
    public People() {                    //无参构造函数
    }
    public String getName() {            //属性 name 的访问器和修改器
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getOld() {                //属性 old 的访问器和修改器
        return old;
    }
    public void setOld(int old) {
        this.old = old;
    }
```

```
    public static void main(String[] args) {
        System.out.println("表示用户信息的简单 JavaBean!");
    }
}
```

【代码解析】

- 所谓的简单属性就是非数组型属性，在简单属性的属性修改器中，仅仅用来设置属性的值，而不用返回任何结果，所以其返回类型为 void。同时还要注意其属性修改器中只接受一个参数，参数的类型由属性的特性决定，例如 public void setName(String name)和 public void setOld(int old);
- 简单属性的属性访问器仅仅用来返回属性的值，所以它不接受任何参数，但它要返回一个值。同时要注意返回值的类型，必须与修改器所接受的参数类型一致，例如 public String getName()和 public void setOld(int old)。本例的运行结果如图 1.75 所示。

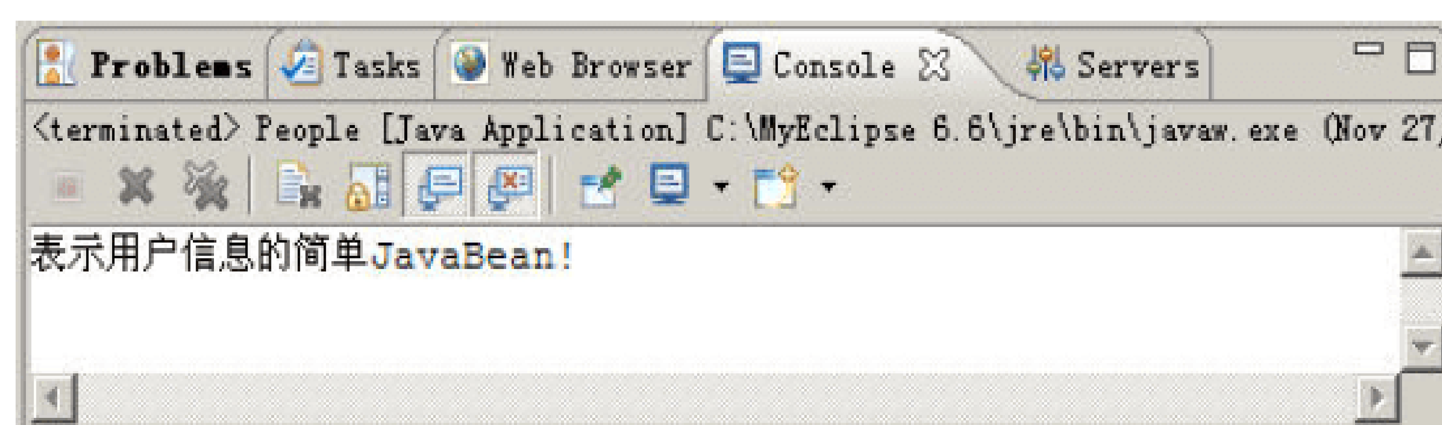


图 1.75　简单属性的运行结果

1.3.7　JavaBean 的属性——复杂属性

复杂属性就是数组类型的属性。复杂类型属性的修改器有两种类型：一个是对整个数组进行赋值；另一个是对数组中的每个元素进行赋值。同理复杂类型属性的访问器也有两种类型：一个是返回整个数组；另一个是返回数组中的某个元素。

代码 1.8 是一个代表用户信息的 JavaBean，在该 JavaBean 中设计了两个属性：表示兴趣的复杂属性 specialities 和表示兴趣的 male 属性。由于一个人的兴趣可以有很多个，所以 specialities 属性应该是数组类型，而 male 属性是布尔型。

代码 1.8　复杂信息：PeopleSpec.java

```
public class PeopleSpec {
    //定义了两个属性 male 和 specialities
    private boolean male;
    private String[] specialities ;
    public PeopleSpec () {                       //无参构造函数
    }
    public boolean isMale() {                    //属性 male 的访问器和修改器
        return male;
    }
    public void setMale(boolean male) {
        this.male = male;
    }
    //属性 getSpecialities 的访问器和修改器
    public String[] getSpecialities() {          //返回整个 Specialities 属性
        return specialities;
```

```
        }
        public void setSpecialities(String[] specialities) {
                                                //为整个 Specialities 属性赋值
            this.specialities = specialities;
        }
        //属性 getSpecialities 的访问器和修改器
        public String getSpecialities(int index) {
                                                //返回 Specialities 属性中某个值
            return specialities[index];
        }
        public void setSpecialities(int index,String specialities) {
                                                //设置 Specialities 属性中某个值
            this.specialities[index] = specialities;
        }
        public static void main(String[] args) {
            System.out.println("表示用户兴趣的复杂 JavaBean!");
        }
}
```

【代码解析】

当属性的类型是布尔型时，访问器名称可以不使用 get 为前缀，而是以 is 前缀代替。如果在 JavaBean 中同时存在 getXXX()和 isXXX()方法，那么该属性的访问器为 isXXX。例如，public boolean isMale()。

在代码中为复杂属性 specialities 分别编写了两个属性访问器和两个属性修改器。对于应用在 JSP 中的 JavaBean 复杂属性，一般不会涉及对其中单个元素进行赋值和读取的情况。本例的运行结果如图 1.76 所示。

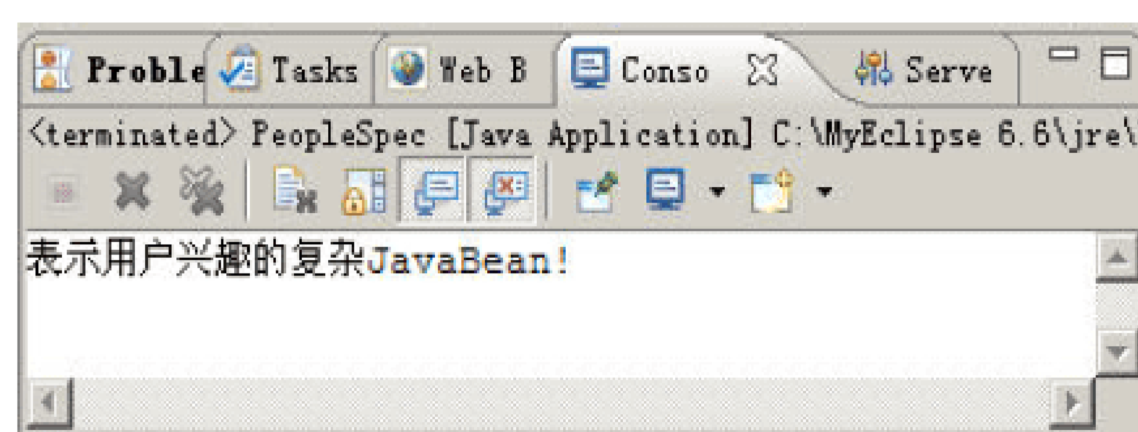


图 1.76　复杂属性的运行结果

1.4　核心框架初步认识

不管开发任何 Java Web 程序，使用基础编程（JSP、Servlet、JavaBean）技术都可以完成，那为什么还要使用框架？之所以使用框架，从根本上来说主要是为了理清程序逻辑和结构，减轻程序员的开发难度，让程序员更加注重业务开发的同时来实现同步开发、提高开发效率和缩短开发周期。

本节将简单介绍一些框架：实现了 MVC 模式的 Structs 框架、无侵入性的 Spring 框架、简单灵活的 Guice 框架、实现持久化的 Hibernate 框架、实现 JPQL 语言的 JPA 框架、实现数据映射器的 iBATIS 框架、用于开发用户界面的 JSF 框架和实现了异步交换的 AJAX 框架。

1.4.1　实现了 MVC 模式的 Structs 框架

在使用 JSP 和 Servlet 开发 Web 应用程序时，可以使用 Servlet 生成 HTML 页面，但是这样所有的代码都必须使用 Servlet 编写；也可以使用 JSP 来生成 HTML 页面，但是业务

逻辑和显示逻辑混杂在一起，代码的维护量大，开发效率低。

为了解决上述的问题，可以使用实现了 MVC 模式的 Structs 框架。Structs 框架是在 Java 服务器端实现了 MVC 设计的模式，其在创建 Java Web 应用程序时能轻易地分离表示层和业务数据层。

什么是 MVC 设计模式呢？MVC 设计模式一般包括 3 个基本部分：Model（模型）、View（视图）和 Controller（控制器），这 3 部分的关系如图 1.77 所示。

- Model：常用来封装和显示数据方面的对象；
- View：可以用来表示数据对象的当前状态，作为模型的显示；
- Controller：用来处理用户的请求并进行转发。

目前最常用的 MVC 框架有 Structs、WebWork，在这些框架中 Structs 不仅在市场上的占有率比较高，同时其拥有的开发人群也是最多的。如果想学好 Structs 框架，那就要从 Structs 1.x 开始。

Structs 1.x 在 2001 年 6 月发布，在当时年代它之所以成功，在于其拥有丰富的文档。该框架以 ActionServlet 作为核心控制器，当用户通过浏览器向服务器发送请求时，请求就会被控制器拦截，然后控制器就会调用模型来处理请求，最后将处理结果通过 JSP 呈现给用户。Structs 1.x 程序运行流程如图 1.78 所示。

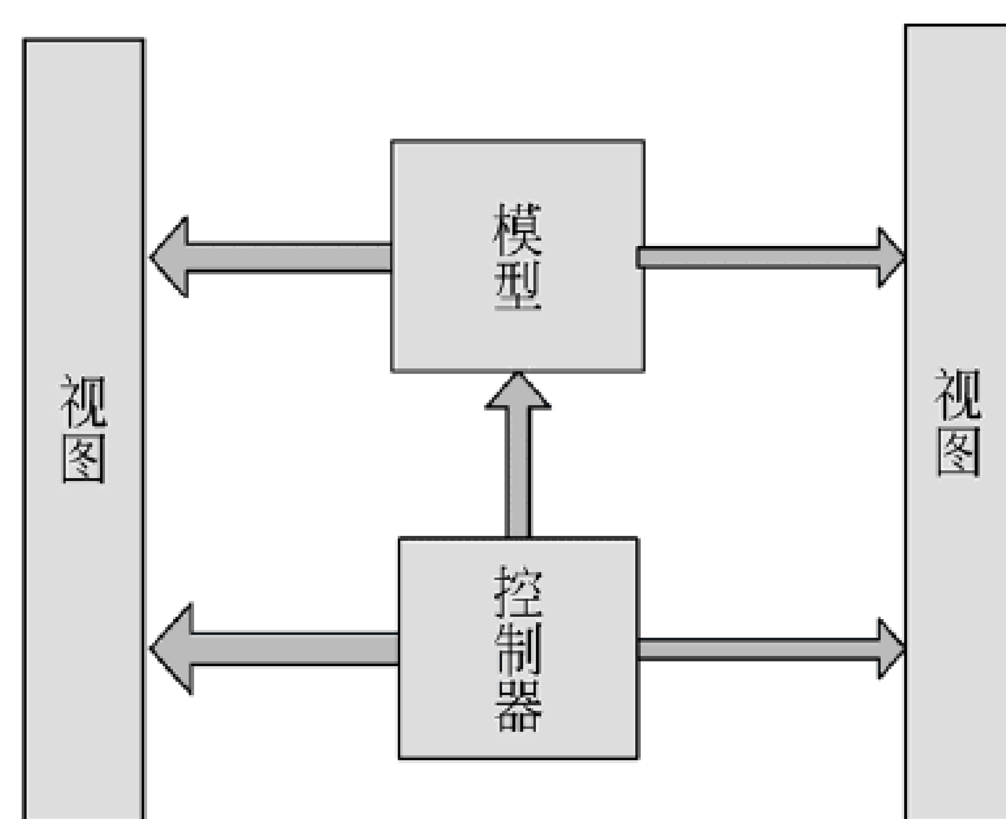


图 1.77　MVC 3 部分关系

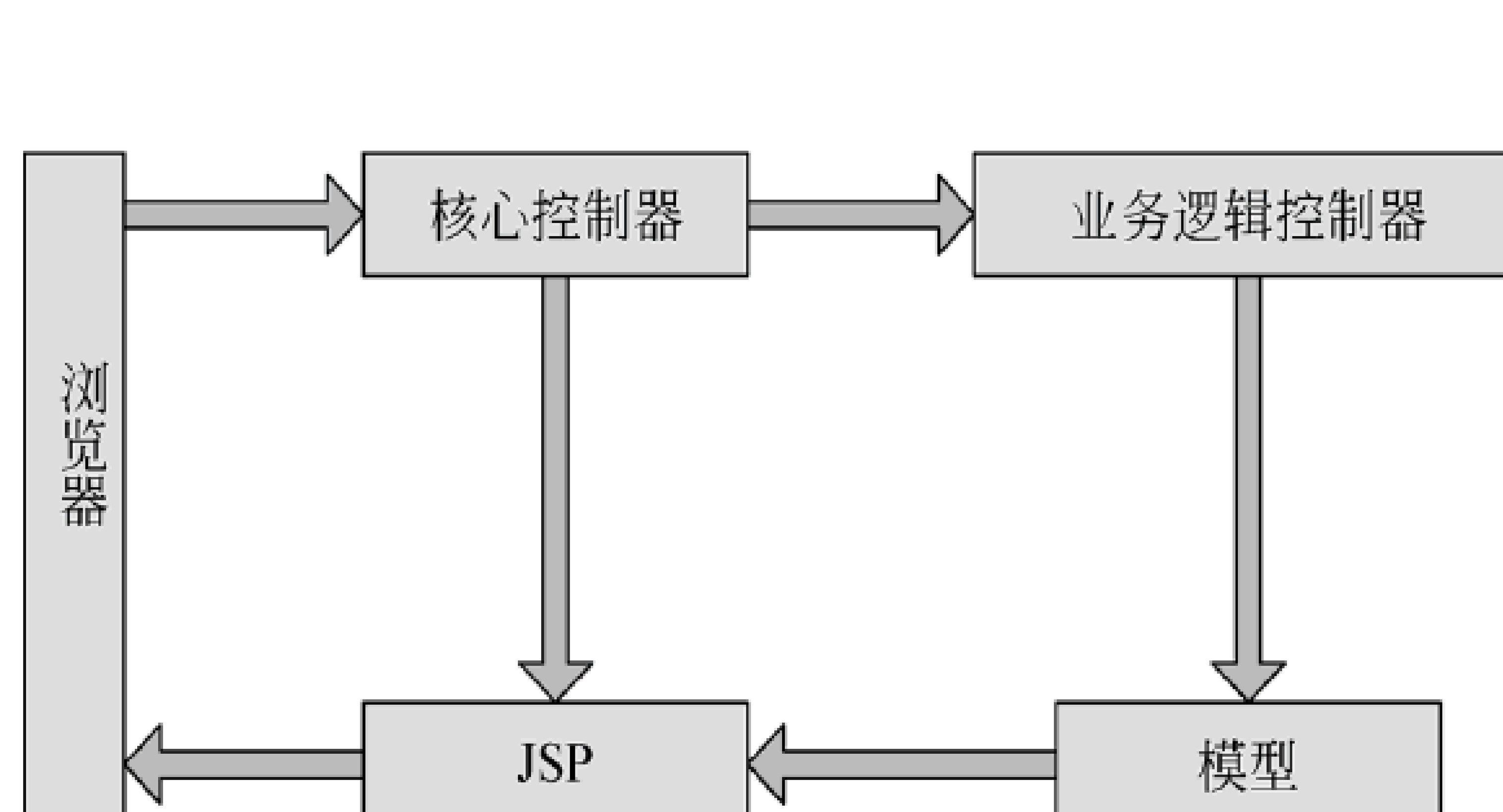


图 1.78　Structs 1.x 的运行流程

在 Structs 1.x 里是这样实现 MVC 中的 3 个角色。

- Model：该部分一般由简单的 JavaBean 来完成底层的业务逻辑组件，但是开发企业级的应用时，就必须使用一个或多个 EJB 组件来完成底层的业务逻辑组件；
- View：该部分采用 JSP 来实现，Structs 1.x 提供了丰富的标签库来支持 JSP 编程；
- Controller：该部分由系统核心控制器和业务逻辑控制器这两个部分组成，系统核心控制器就是系统中的 ActionServlet，而业务逻辑控制器就是用户自己实现的 Action 实例。

注意：Structs 1.x 框架其实只实现了控制器部分，而其他两个部分，该框架只是利用控制器调用模型和把处理结果传送给视图。

随着时间的发展，Structs 1.x 框架也显示出自己的种种缺陷：

- 该框架属于入侵式设计，即使用其实现的程序严重依赖于 Structs 1.x API；
- 该框架的业务逻辑控制器用 Servlet API 来编写，严重依赖于 Web 服务器；
- 该框架的表示层只能使用 JSP 技术来编写，而不能使用 FreeMarker、Velocity 等视

图技术。

经过 5 年的发展，终于出现了 Structs 2.x 框架。Structs 2.x 框架与 Structs 1.x 框架差别非常大，为了将业务逻辑控制器与 Servlet API 分离，该框架使用大量的拦截器来处理用户请求。同时 Structs 2.x 框架的表示层除了支持 JSP，同时也支持 FreeMarker、Velocity 等视图技术。Structs 2.x 框架的运行流程如图 1.79 所示。

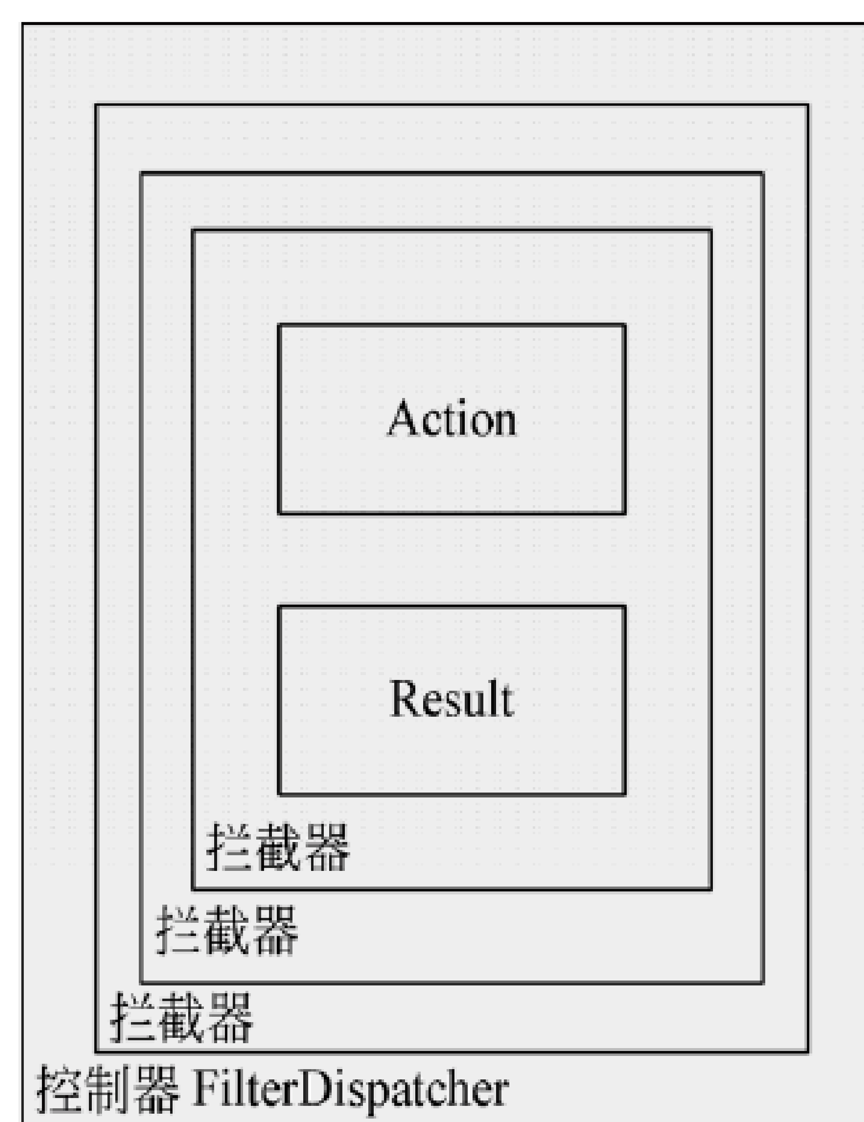


图 1.79　Structs 2.x 的运行流程

当用户通过浏览器向服务器发送请求时，请求就会被核心控制器 FilterDispatcher 拦截，然后核心控制器就会调用合适的 Action 来处理请求。在调用 Action 的过程中请求会经过多层拦截器，这些连接器会对请求应用通用功能。最后回调 Action 的 execute()方法，把处理好的信息传送给浏览器。

1.4.2　无侵入性的 Spring 框架

Spring 在英文中是“春天”的意思，春天意味着万物复苏，而 Spring 框架也将担当此任。当使用 Spring 框架时，它会使开发者和具体 J2EE 平台技术处于“松耦合”的状态。对于各种 Java Web 框架来说，Spring 框架就是黏合剂。假如 Structs 框架负责表单提交相当于电脑显卡，Hibernate 框架负责对数据库的操作相当于计算机的 CPU，那么 Spring 框架相当于一个主板将显卡和 CPU 组装在一起。

Spring 框架是一个分层架构，其主要特性被很好地组织在 7 个模块中，如图 1.80 所示。如果想学好 Spring 框架，就要尽可能掌握各个模块。

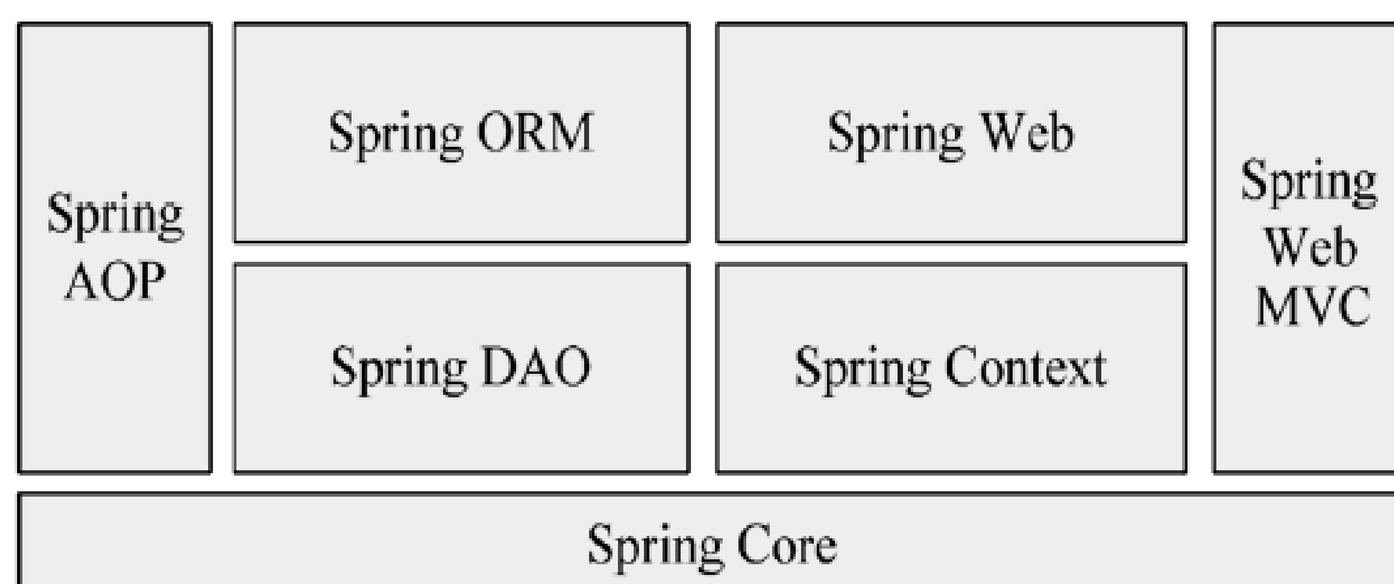


图 1.80　Spring 框架模块

- Spring Core（Spring 核心容器）：它是 Spring 框架的最基础部分，对该部分要理解 3 个概念：IOC（控制反转）、DI（依赖注入）和 BeanFactory（Bean 工厂）。IOC 代表的是一种思想，也是一种开发模式，是解决调用者（Bean）和被调用者（Bean）之间的一种关系。使用 IOC 不仅使应用中对象的关系清晰、一致，而且还使一切对象可控。DI（依赖注入）就是 IOC 的实现策略，该策略让容器全权负责依赖，受控对象只要暴露属性和带参数的构造函数，在初始化对象的时候就可以设置对象间的依赖关系。BeanFactory 利用经典的 Factory 模式来消除对程序性单例的需要，并允许从程序逻辑中分离出依赖关系的配置和描述；
- Spring Context（Spring 上下文）：是一个配置文件，可以向 Spring 框架提供上下

文信息。该框架利用一种框架式的对象方法来实现一些企业服务，例如 JNDI、EJB、电子邮件、国际化、校验和调度功能；

- Spring DAO（Spring 数据访问操作）：是 JDBC 的抽象层，利用异常层次结构消除冗长的 JDBC 编码和解析不同数据库厂商特有的错误代码。同时也提供了一种针对所有操作数据库类的方法来实现编程性和声明性事务管理；
- Spring AOP（Spring 面向切面编程）：是 Spring 的拦截器，借助于面向切面编程可以以声明式的方式使用企业级服务。所谓 AOP 就是允许定义方法拦截器和切点，来干净地给从逻辑上说应该被分离的功能实现代码解耦。借助于 Spring AOP，开发者可以声明式、基于元数据访问企业级服务；
- Spring ORM（Spring 对象关系映射）：ORM 就是流行的关系——对象映射，该包为 Spring 框架提供了关系——对象映射工具，例如 JDO、Hibernate 和 iBATIS SQL Map。同时 Spring 框架也为关系——对象映射提供了一些特性，例如通用事务和 DAO 异常层次结构；
- Spring Web（Spring 的 Web 上下文）：是 Spring 的 Web 上下文，与 Spring 上下文的区别在于前者是构建在 Spring Core 基础上，而后者是构建在应用程序基础上。该模块的主要作用是为基于 Web 的应用程序提供上下文，例如多方文件上传、Multipart 功能、使用 Servlet 监听器的 Context 的初始化和面向 Web 的 Applicatin Contex，以及使 Spring 框架可与其他框架结合；
- Spring Web MVC（Spring Web 应用程序的 MVC）：是面向 Web 应用的 MVC 实现。Spring Web MVC 不但使传统 MVC 框架成为高度可配置，而且还提供了一种清晰分离模式使 MVC 框架可使用 Spring 框架的所有其他特性，如校验等。

1.4.3　简单灵活的 Guice 框架

在过去是以面向接口的编程方式来开发普遍的应用，但是这种方式需要程序员自己处理接口和这些接口实现类之间的关系，以及访问中间层和事务管理器的操作。为了解决该弊端，于是就出现了在 IOC 框架中使用 XML 配置文件进行 Bean 配置的解决方案。但是该方案无法把代码修改和配置文件的修改同步，同时配置文件也无法进行类型检测。随着时间的发展，Spring 和 Guice 这两种框架都可以解决上述问题。

Spring 和 Guice 这两种框架虽然都可以实现依赖注入和配置，但是 Guice 框架和 Spring 框架之间最主要的区别，可以归结为它们实现依赖关系和配置理念。Spring 框架推崇主要的是非侵入性的方式，即以一种完全外部化的方式来对待对象依赖关系。例如，可以使用 XML、Spring JavaConfig、Groovy-Spring DSL 和 Spring-annotations 等外部技术来实现对象的依赖关系。Guice 框架却把配置作为应用程序模型的首要对象来看待，允许它们存在于领域模型代码中。

下面通过斧子的例子来详细讲解实现依赖关系的各种方式，它们分别如下。

（1）原始社会时，劳动社会基本没有分工，需要斧子的人（调用者）只好自己去磨一把斧子，每个人拥有自己的斧子。如果把大家的石斧改为铁斧，需要每个人都要学会磨铁斧的本领，工作效率极低。

【对应 Java 代码中的情形是】当调用者需要被调用者时，都必须得通过 new 来实现。类耦合度极高，修改维护繁琐，效率极低。

（2）工业社会时，工厂出现，斧子不再由普通人完成，而由工厂生产，当人们需要斧子的时候，可以到工厂购买斧子，无须关心斧子是怎么制造出来的。如果废弃铁斧为钢斧，只需改变工厂的制造工艺即可，制造工艺是工厂决定的，工厂生产什么斧子，工人们就得用什么斧子。

【对应的 Java 中的情形是】当调用者需要被调用者时，可以使用工厂模式来创建被调用者。由于变化东西被封装到工厂，所以耦合度降低，但是调用者还是会和工厂耦合。

（3）近代工业社会，工厂蓬勃发展，人们需要什么斧子，只需要提供一个斧子图形，商家会按照你提供的图形将你的斧子订做好，并送上门。

【对应 Java 中的情形】Spring 框架的依赖注入。

（4）进入按需分配社会，信息进入现代化，人们不再去工厂购买斧子，不再拘泥于需要什么斧子事先画好什么样的图形，只需要打个电话，描述一下需要什么类型的斧子，或许想打造一个物美价廉的斧子，商家会根据市场零件的价格，计算出最优制作工艺，打造最适合的斧子送过来，更加信息化，更加人性化。

【对应 Java 中的情形】Guice 框架的依赖注入，即基于描述的注入，动态的、灵活简单的注入。

通过上述的分析可以发现，与 Spring 框架相比，Guice 框架拥有更加强大的功能，并且 Structs 2.x 框架中名为 xwork 2.0.jar 包已经集成了该框架。

1.4.4　实现持久化的 Hibernate 框架

关系数据库的操作是基于行集，因此出现各种方式的 ODBC、BDE、ADO、ADO.NET 和 JDBC 的数据库操作也是基于行集。在编写具体应用程序时，应用程序承担着从行集中获取数据来展示的工作，这种工作让程序员不断重复相同的枯燥无味的过程。随着面向对象技术的流行，一场新的革命终于来临，即程序员开始了对象与数据管理结合——对象/关系数据库映射（ORM）。所谓 ORM 就是把对象模型表示的对象映射到基于 SQL 的关系模型结构，在目前最完善和最强悍的 ORM 产品就是 Hibernate 框架。

在大多数企业级应用中，数据持久化意味着将内存中的数据保存到磁盘上，而该过程的实现大多数是通过各种关系型数据库来完成。那数据持久层是什么？在开发项目时经常会提到三层结构，其构成如图 1.81 所示。但是在具体项目的实际开发当中，设计者通常对三层结构扩展成五层结构来满足项目的具体要求，如图 1.82 所示。数据持久层位于领域层和基础架构层之间，用来在“表格型的关系型数据库与树型的程序对象”之间提供一个映射解决方案——ORM。

如何实现数据持久层，在学术界存在两种方向。

- ❑ 直接编写 JDBC 等 SQL 语句（如 iBATIS）；
- ❑ 使用 O/R Mapping 技术实现的 Hibernate。

Hibernate 框架不仅能够管理 Java 类到数据库表的映射，还提供了数据查询和获取数据的方法。该框架通过支持数据持久化相关的编程来减少开发者人工使用 SQL 和 JDBC 处理

数据的时间。Hibernate 框架的体系结构如图 1.83 所示。



图 1.81　三层结构　　图 1.82　五层结构　　图 1.83　Hibernate 框架的体系结构

Hibernate 框架存在一些在任何开发中都会用到的核心接口，它们分别为：

- Session 接口：Session 接口负责执行被持久化对象的 CRUD 操作（CRUD 的任务是完成与数据库的交流，包含了很多常见的 SQL 语句）。但需要注意的是 Session 对象是非线程安全的。同时，Hibernate 的 Session 不同于 JSP 应用中的 HttpSession。这里当使用 Session 这个术语时，其实指的是 Hibernate 中的 Session，而以后会将 HttpSesion 对象称为用户 Session；
- SessionFactory 接口：SessionFactroy 接口负责初始化 Hibernate。它充当数据存储源的代理，并负责创建 Session 对象。这里用到了工厂模式。需要注意的是 SessionFactory 并不是轻量级的，因为一般情况下，一个项目通常只需要一个 SessionFactory 就够了，当需要操作多个数据库时，可以为每个数据库指定一个 SessionFactory；
- Configuration 接口：Configuration 接口负责配置并启动 Hibernate，创建 SessionFactory 对象。在 Hibernate 的启动的过程中，Configuration 类的实例首先定位映射文档位置、读取配置，然后创建 SessionFactory 对象；
- Transaction 接口：Transaction 接口负责事务相关的操作。它是可选的，开发人员也可以设计编写自己的底层事务处理代码；
- Query 和 Criteria 接口：Query 和 Criteria 接口负责执行各种数据库查询。它可以使用 HQL 语言或 SQL 语句两种表达方式。

1.4.5　实现 JPQL 语言的 JPA 框架

JPA 框架全称 Java Persistence API，为了得到所有 Java EE 服务器的支持，该框架实现 Java EE 5.0 平台标准的 ORM 规范。JPA 框架是 Sun 公司在吸取了之前 EJB 规范惨痛失败的经验后而出现的新框架。从目前开发人员的反应上看，JPA 受到了极大的支持和赞扬。

实现持久层的技术很多，像 1.4.4 节介绍的 Hibernate 框架和 1.4.6 节将要介绍的 iBATIS 框架技术。然而作为 EJB 3.0 组成部分，JPA 框架是这样定义的：JPA 是一个基于操作持久层的框架，它使用 ORM 映射将面向模型和关系数据库联系起来。

总体来说，JPA 框架由 3 部分组成，分别如下。

- Java 持久化 API：通过操作实体对象来对数据库中的数据进行增、删、改、查（CRUD）和事物的操作，该框架在后台替程序员完成所有的事情，使得程序员从繁琐的 JDBC 和 SQL 代码中解脱出来；
- ORM（对象/关系）映射元数据：JPA 框架支持 XML 和 JDK 5.0 注解两种元数据描述形式，通过使用元数据描述对象和表之间的映射关系，从而将实体对象持久化到数据库表中；
- 查询语言：作为持久化操作中很重要的一个方面，JPA 框架通过面向对象而不是面向数据库的查询语言查询数据，避免程序中 SQL 语句的紧密耦合。

与其他持久层框架相比较，JPA 框架具有如下的特点：

- 在 JPA 框架中建立一个数据库实体类非常简单，只需要在一个普通的 Java 类中加入一些相应的标注就可以使该类由简单类变成实体类；
- 在 JPA 框架中如果想操作数据库，可以通过调用实体管理器（EntityManager）中的方法对实体类进行相应的操作；
- 在 JPA 框架中如果想实现事务处理，可以通过调用实体管理器（EntityManager）中的 getTransaction()方法获取事务，从而进行相应的操作；
- 在 JPA 框架中支持名为（Java Persistence Query Language，JPQL）的查询语句，该查询语句是从 Hibernate 框架中的 HQL 语句继承过来的。JPQL 语句不仅支持批量更新、JOIN、GROUPBY、HAVING 等 SQL 语句中的高级查询，而且还支持子查询。

1.4.6　实现数据映射器的 iBATIS 框架

iBATIS 单词是由 internet 和 abatis 两个单词组合而成，该单词实际上就是通常所说的数据映射器（data mapper）。iBATIS 框架的开发教父 Martin Fowler 在他自己的《Patterns of Enterprise Application Architecture》一书中这样描述数据映射器模式：所谓映射器，是用于在对象和数据库之间搬运数据，同时保证对象、数据库以及映射器本身都相互独立。

以数据映射为核心的 iBATIS 框架，实际上是基于 JDBC 之上的一层简单抽象，实现将数据库表列映射到领域模型层对象的功能。同时该框架也是以 SQL 作为持久化数据语言的 ORM 的解决方案。

在具体使用 iBATIS 框架时，该框架通过配置文件加载数据源、映射字段与 Java 对象的属性来完成数据的持久化。通常需要两个配置文件，一个用来配置 iBATIS 框架自身的一些属性信息、缓冲机制、事务处理方式及数据源；另一个是映射文件，将数据表映射到领域模型对象。

在 iBATIS 框架中有一个核心组件 SQL Map 用来实现数据映射，在 Hibernate 框架中通过对象/关系映射（O/RM）工具来实现元数据映射。那么元数据映射和数据映射有什么区别呢？

元数据映射实际上是将数据库表及其列映射为应用中的类及字段，即在数据库的元数据与类的元数据之间建立起了一种映射关系。图 1.84 展示了所谓的元数据映射，它在一个类与数据库表之间建立了映射关系。在这种情况下，类的每一个字段都被映射为数据库中

相应表的唯一列。

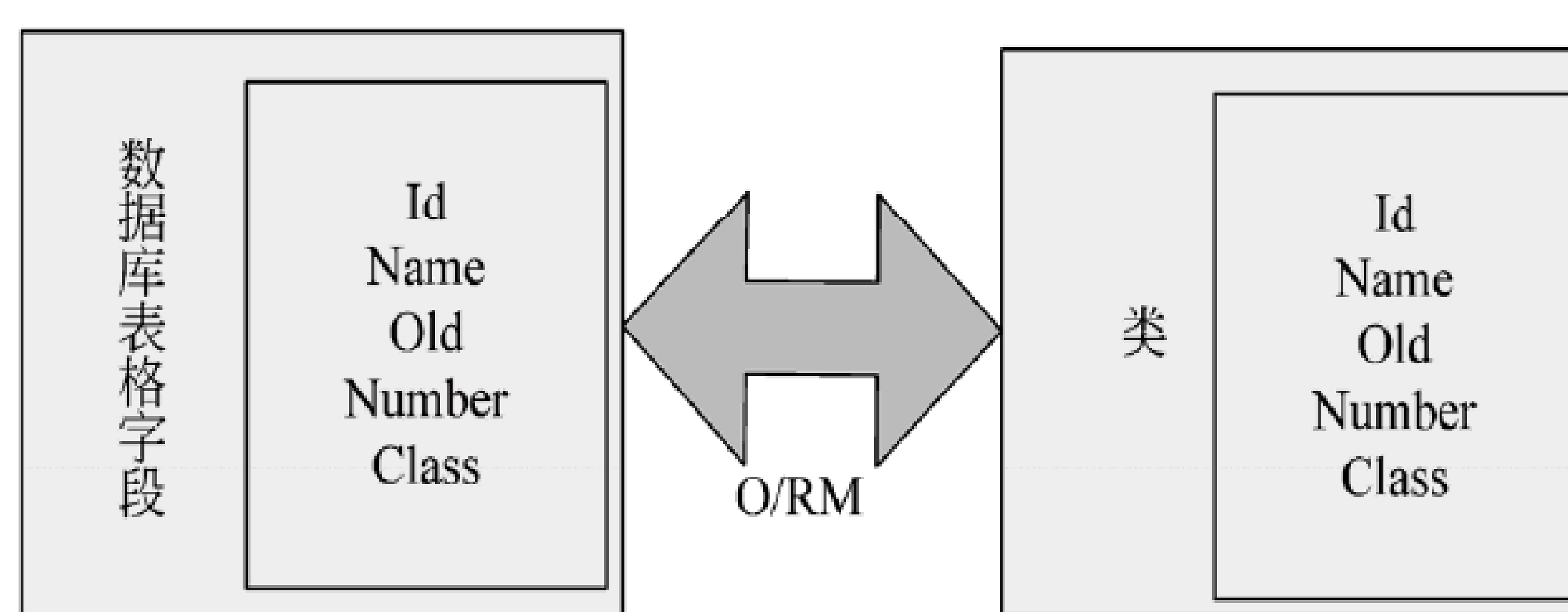


图 1.84　元数据映射

数据映射不是直接把类映射为数据库表或者说把类的字段映射为数据库列，而是把 SQL 语句的参数与结果（也即输入和输出）映射为类，如图 1.85 所示。即在类和数据库表之间建立了一个额外的间接层，这就为在类和数据库表之间建立映射关系带来了更大的灵活性，使得在不用改变数据模型或者对象模型的情况下可以改变它们的映射关系。所谓的间接层就是 SQL Map，该层不仅在对象和数据库间传递数据，而且还保持两者与映射层本身相独立。

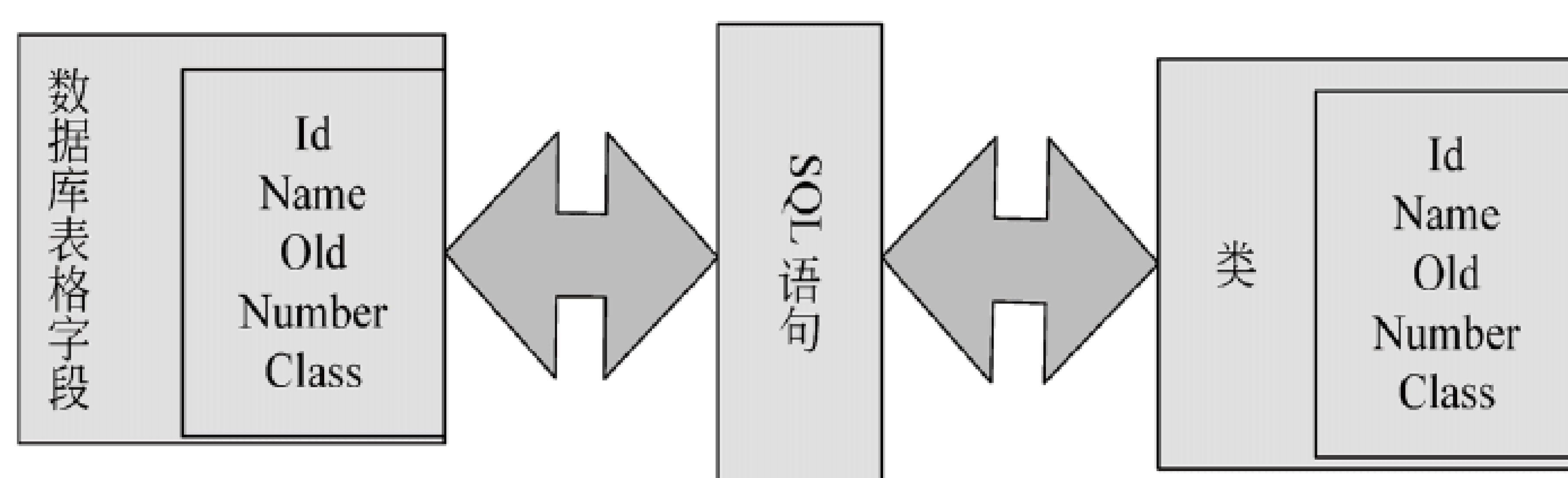


图 1.85　iBATIS 的 SQL 映射

在具体编写时，程序员只需负责编写 SQL 语句，而 iBATIS 框架负责在类的字段和数据库表的列之间映射参数和结果。

1.4.7　用于开发服务器端用户界面的 JSF 框架

在开发 Java Web 方面的运用程序时，可以使用基础 JSP 技术来实现各层功能，这既是 JSP 技术的最大优点同时也是它的最大缺点。因为功能强大就意味着复杂，特别是在利用 JSP 来实现显示层，可利用的可视化工具不仅不多，而且功能还不够强大。Sun 为了简化 JSP 的开发难度，推出了新的框架技术——JavaServer Faces（JSF）框架，该框架主要用来开发 Java Web 应用程序的服务器端用户界面。

从网页设计人员的角度来看，JSF 提供了一套标准动态标签。这些标签不同于 HTML 标签，可以与后端的动态程式结合，在具体设计网页时网页设计人员不需要理解后端的动态部分，甚至不太需要了解动态标签，就可以实现动态的显示。

注意： JSF 提供的标准标签不仅可以与网页编辑程式结合在一起，而且还可以允许网页设计人员自己定制。

从应用程式设计人员的角度来看，JSF 提供一个基于事件驱动的页面导航模型。通过

该模型开发程序时，应用程式设计人员不必关切 HTTP 的处理细节，就能够设计应用程序的页面流。

注意：JSF 框架虽然比其他 Web 框架复杂，但是对应用程式设计人员却隐藏了其复杂性。即开发人员只要处理 Java 代码就可以，而不需处理请求对象、会话对象、请求参数等。

从 UI 元件开发人员的角度来看，作为标准的技术，JSF 可以设计通用的 UI 元件。当 UI 开发人员开发通用 UI 元件时，只要定义好该元件的属性选项就可以，而不必受到网页设计人员或应用程式设计人员的干扰。

注意：JSF 与桌面的 UI 框架如 Swing 或 SWT 之间的不同之处是：前者是运行在服务器端，而后者是运行在一个标准的 Java Web 容器上。

JSF 框架除了可以开发服务器端用户界面，同时其还严格遵循了 MVC 设计模式的框架。JSF 的 MVC 实现如图 1.86 所示。

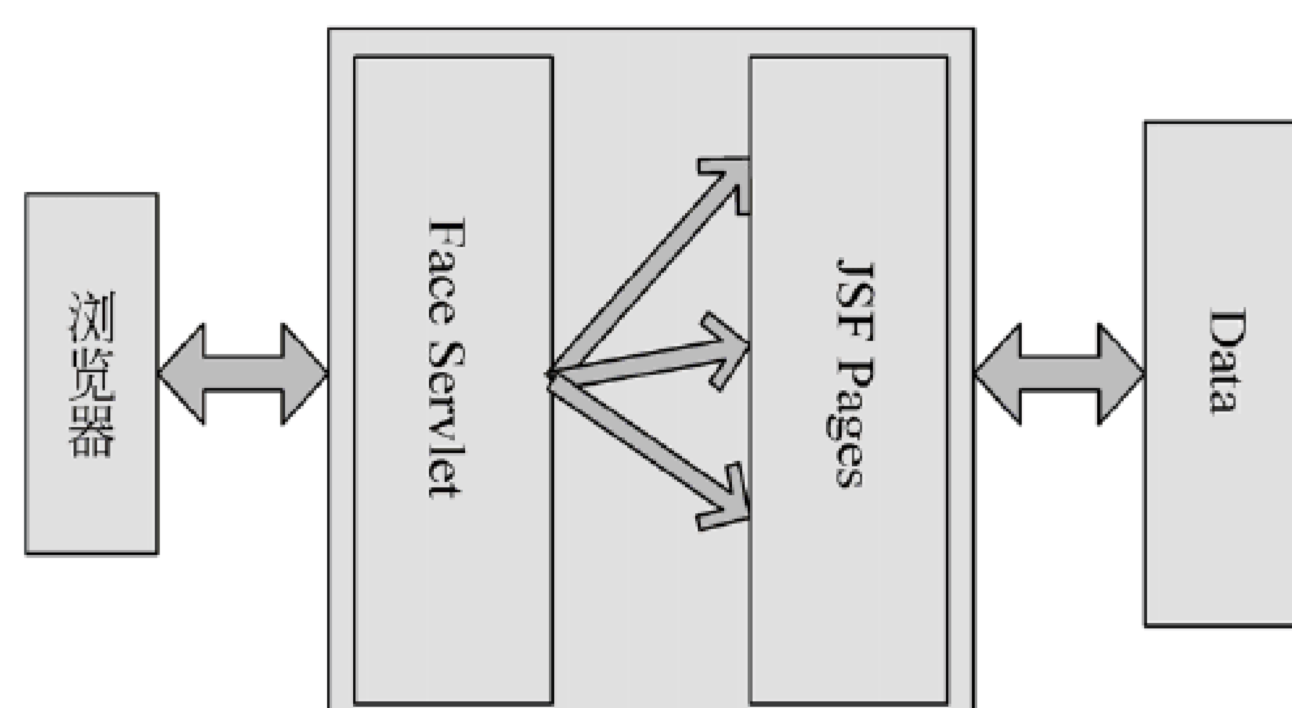


图 1.86　JSF 的 MVC 实现

JSF 应用程序为了便于管理，一般都是把用户界面代码（View）与应用程序数据和逻辑（Model）分离，而它们之间的交互则由一个前端 Faces Servlet（Controller）来处理。当用户通过浏览器发送一个请求到服务器时，该请求会被 Faces Servlet 转换成一个可以被服务器中的应用逻辑处理的事件，同时其也负责保证在服务器所定义的每一个 UI 部件都正确显示给浏览器。

1.4.8　实现了异步交换的 AJAX 框架

为什么会出现 AJAX？当浏览器向 Web 服务器发送一个请求时，服务器接收并处理传来的表单，然后返回一个新的网页。这个做法浪费了许多带宽，因为在前后两个页面中的大部分 HTML 代码往往是相同的。由于每次应用的交互都需要向服务器发送请求，应用的响应时间就依赖于服务器的响应时间。这导致了用户界面的响应比本地应用慢得多。

所以就出现了 AJAX，这种技术不仅向服务器发送并取回必需的数据，而且它使用 SOAP 或其他一些基于 XML 的 Web Service 接口，在客户端采用 JavaScript 处理来自服务器的响应。因此在服务器和浏览器之间交换的数据大量减少，结果就能很快看到响应。同时很多的处理工作可以在发出请求的客户端机器上完成，所以 Web 服务器的处理时间也减

少了。

坦率地讲，AJAX 并不是什么新的编程语言。实际上，与这个词相关的“最新”术语就是 XMLHttpRequest 对象（XHR），它早在 IE 5（于 1999 年春天发布）中就已经出现了，是作为 Active X 控件露面的。

AJAX 技术在大多数现代浏览器中都能使用，而且不需要任何专门的软件或硬件。实际上，这种方法的一大优势就是开发人员不需要学习一种新的语言，也不必完全丢掉它们原先掌握的服务器端技术。AJAX 是一种客户端方法，它并不关心服务器是什么。尽管存在一些很小的安全限制，但是开发人员却可以利用原有的知识，马上就能使用 AJAX 技术。

AJAX 是由 HTML、JavaScript 技术、DHTML 和 DOM 组成，它不仅仅是一种时尚，也是一种构建网站的强大方法，而且不像学习一种全新的语言那样困难。它所包含的技术有：

- HTML 用于建立 Web 表单并确定应用程序其他部分使用的字段；
- JavaScript 代码是运行 AJAX 应用程序的核心代码，帮助改进与服务器应用程序的通信。使用 XMLHttpRequest 来和服务器进行异步通信；
- DHTML 或 Dynamic HTML，用于动态更新表单。我们将使用 div、span 和其他动态 HTML 元素来标记 HTML；
- 文档对象模型 DOM 用于（通过 JavaScript 代码）处理 HTML 结构和（某些情况下）服务器返回的 XML；

AJAX 应用程序的优势在于以下 3 方面。

- 通过异步模式，提升了用户体验；
- 优化了浏览器和服务器之间的传输，减少不必要的数据往返，减少了带宽占用；
- AJAX 引擎在客户端运行，承担了一部分本来由服务器承担的工作，从而减少了大用户量下的服务器负载。

1.5　小　　结

本章首先讲解了开发 Java Web 项目的基本知识和所需的基础软件：JDK、Tomcat、MyEcllipse、MySQL 和 Oracle，通过对这些基础软件下载和安装方面的详细讲解，使读者能够很容易实现关于 Java Web 项目的开发环境。接着通过 1.3 和 1.4 两节的内容简单介绍了本书将要使用的框架：JSP+Servlert+JavaBean、Structs、Spring、Guice、Hibernate、JPA、iBATIS、JSF 和 AJAX。

第 2 章　MyEclipse 开发工具对各种框架的支持

开发一个具体的软件项目之前，最重要的就是要决定使用什么解决方案，所谓解决方案就是选择什么框架技术来开发项目。其次还需要为软件项目进行划分，即首先需要把整个软件项目划分成不同模块，然后对每个模块进行分层。

本章将详细讲解如何通过框架和它们的集成来实现软件项目，同时由于每个 Java Web 方面的软件项目都涉及数据库方面的操作，所以还将简单介绍如何使用 JDBC 连接数据库。涉及的框架如下：

- JSP+JavaBean 和 Servlet+JSP+JavaBean 框架；
- Struts 框架；
- Hibernate 框架；
- JPA 框架；
- Spring 框架；
- JSF 框架。

2.1　使用 JSP 的两种模式

Sun 公司在 JSP 规范中定义了两种模式：

- Model 1（模式一）：JSP+JavaBean 技术；
- Model 2（模式二）：Servlet+JSP+JavaBean 技术。

这些模式的使用可以很好地利用 JSP 来开发 Java Web 应用程序，本节将详细介绍这两种模式。

2.1.1　开发环境 MyEclipse 对模式 1 的支持

MyEclipse 开发环境对 Model 1 的开发提供了全方位的支持，即可以用向导的方式创建 JSP，同时也对 JavaBean 的编写提供了支持。对利用 Model1 模式来编写大型网站的程序开发者来说，MyEclipse 是非常实用和方便的。下面将详细介绍如何实现 Model 1，具体步骤如下。

（1）首先创建 Web 项目，通过选择 File|New|Web Project 命令，启动创建 Web 项目向导，如图 2.1 所示。

在 Web 项目详情（Web Project Details）中，各项的具体意义如下。

- Project Name：项目名称。
- Location：项目文件存储的位置。
- Source folder：源代码的目录。
- Web root folder：该目录包含 Web 应用的内容，WEB-INF 目录以及对应的子目录。
- Context root URL：Web 项目在发布时所使用的路径。

J2EE Specification Level 用来指定 J2EE 规范的版本，而 JSTL Support 用来指定 JSTL 的版本。

（2）接着创建 JSP 页面，通过选择 File|New|JSP 命令启动创建 JSP 页面向导，如图 2.2 所示。

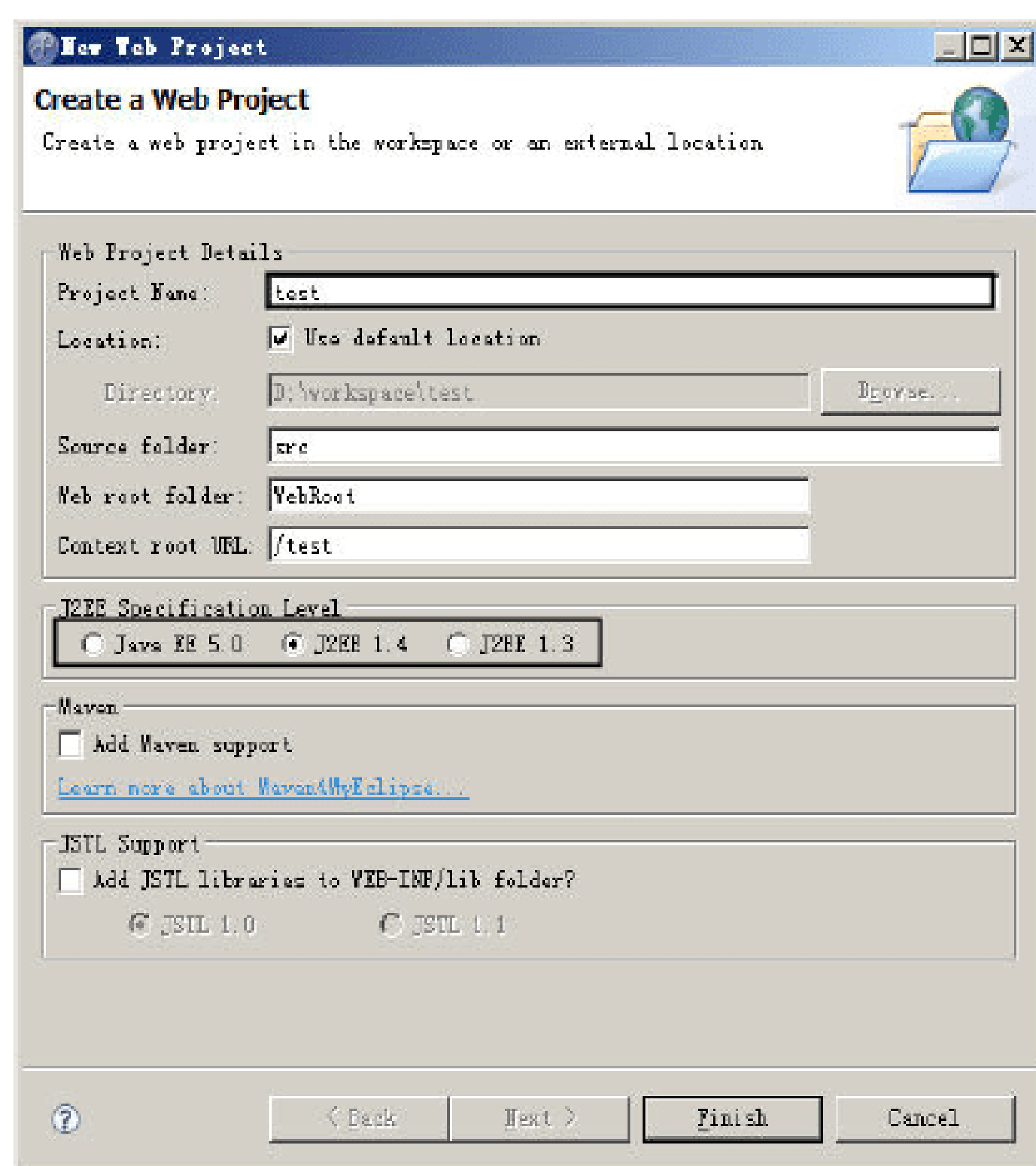


图 2.1　创建 Web 项目

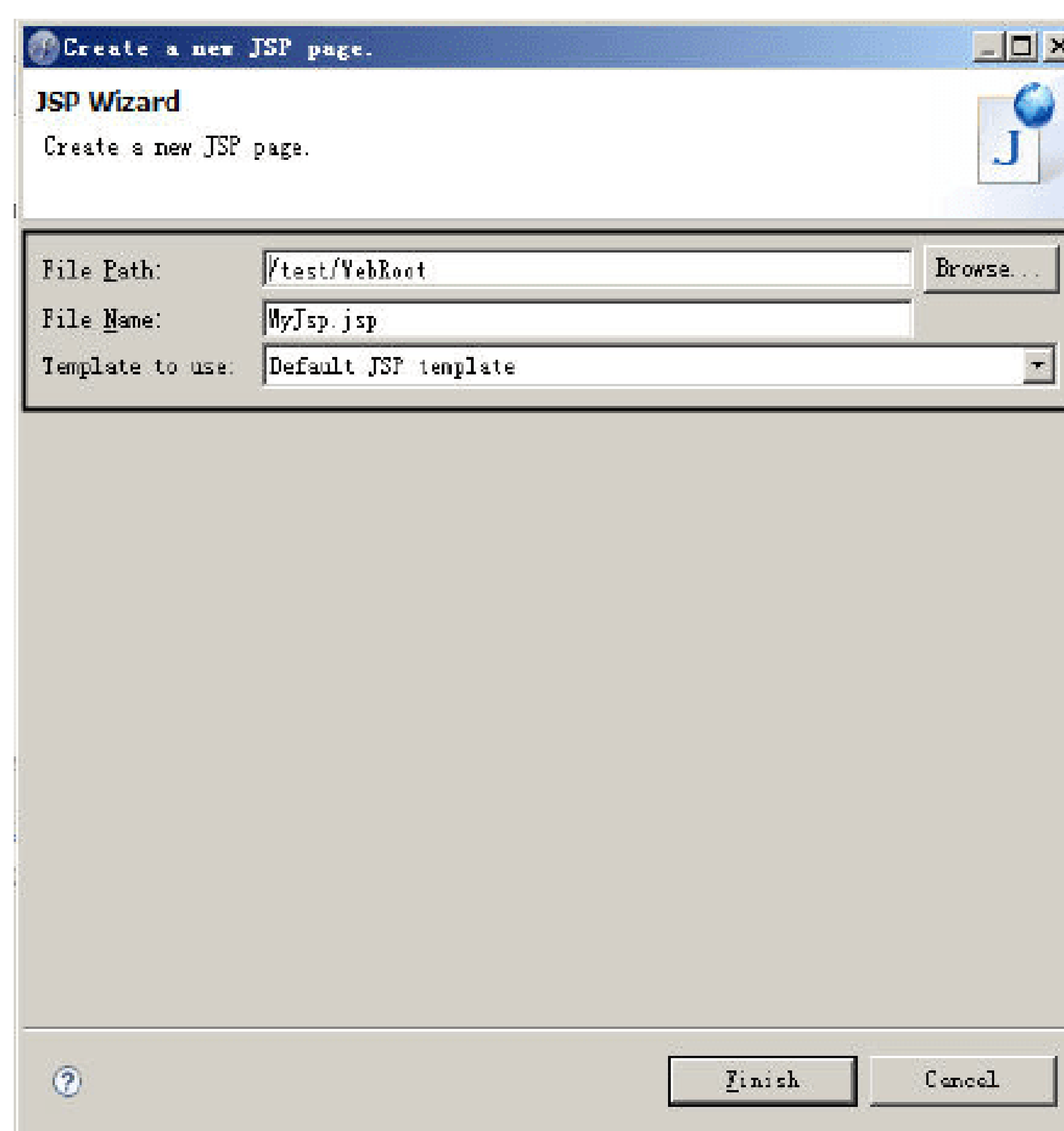


图 2.2　创建 JSP 页面

在创建一个新的 JSP 文件中，各个选项的具体含义如下。

- File Path：文件的目录。
- File Name：文件的名字。
- Template to use：JSP 文件的模板。

（3）最后创建 JavaBean 代码，通过选择 File|New|Class 命令，启动创建 Class 向导。然后在具体代码中编写一个成员变量，接着右击该变量，在弹出的快捷菜单中选择 Source|Generate Getters and Setters 命令就会打开 Generate getters and setters 对话框，如图 2.3 所示。在该对话框的 Select getters and setters to create 选项中选择要生成的方法，然后单击 OK 按钮就会自动生成 get()和 set()方法。

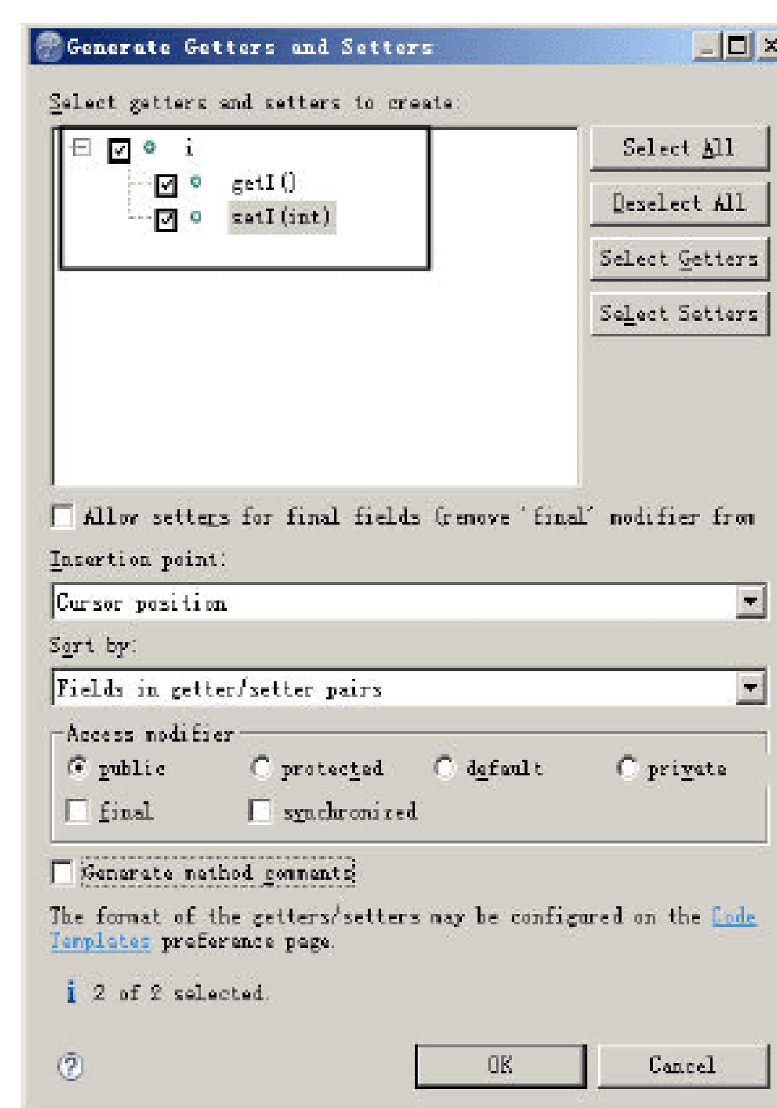


图 2.3　生成 get()和 set()方法

经过上述 3 个步骤后，该项目就拥有了 Model 1 模式的各个组件部分。

2.1.2　开发环境 MyEclipse 对模式 2 的支持

MyEclipse 开发环境对 Model 2 的开发提供了全方位的支持，既可以用向导的方式创建

JSP 和 Servlet，同时也对 JavaBean 的编写提供了支持。对利用 Model 2 模式来编写大型网站的程序开发者来说，MyEclipse 是非常实用和方便的。下面将详细介绍如何实现 Model 2，具体步骤如下。

（1）首先需要创建一个 Java Web 项目，接着为该项目添加 JSP 和 JavaBean 组件。由于这些组件在上述章节已经介绍，所以不再详细介绍。

（2）与 Model 1 相比，Model 2 模式中多出了 Servlet 组件。如果想创建 Servlet 程序，可以通过菜单 File|New|Servlet 命令启动创建 Servlet 向导，如图 2.4 所示。

在创建一个新的 Servlet 程序中，各个选项的具体含义如下。

- Source folder：源代码目录。
- Package：设置包。
- Name：Servlet 程序的名字。
- Modifiers：Servlet 程序的修饰符。
- Superclass：Servlet 程序的父类
- Interfaces：Servlet 程序的接口。
- Template to use：Servlet 文件的模板。
- Which method stubs would you like to create？：Servlet 程序继承的方法。

当单击 Next 按钮后，就会进入修改、设置 web.xml 的向导页面，如图 2.5 所示。在该对话框中一般只需修改两个地方：Servlet/JSP Name 和 Servlet/JSP Mapping URL，前者用来设置 Servlet 程序的注册名字，后者用来设置 Servlet 程序的映射路径。

注意：如果想访问 Servlet 程序，必须通过该 Servlet 程序的映射路径来访问而不是其名字。同时还要注意 Servlet 程序的映射路径一定要以/开始，或者以*.do 的形式出现，但是这种方式/*do 是错误的。

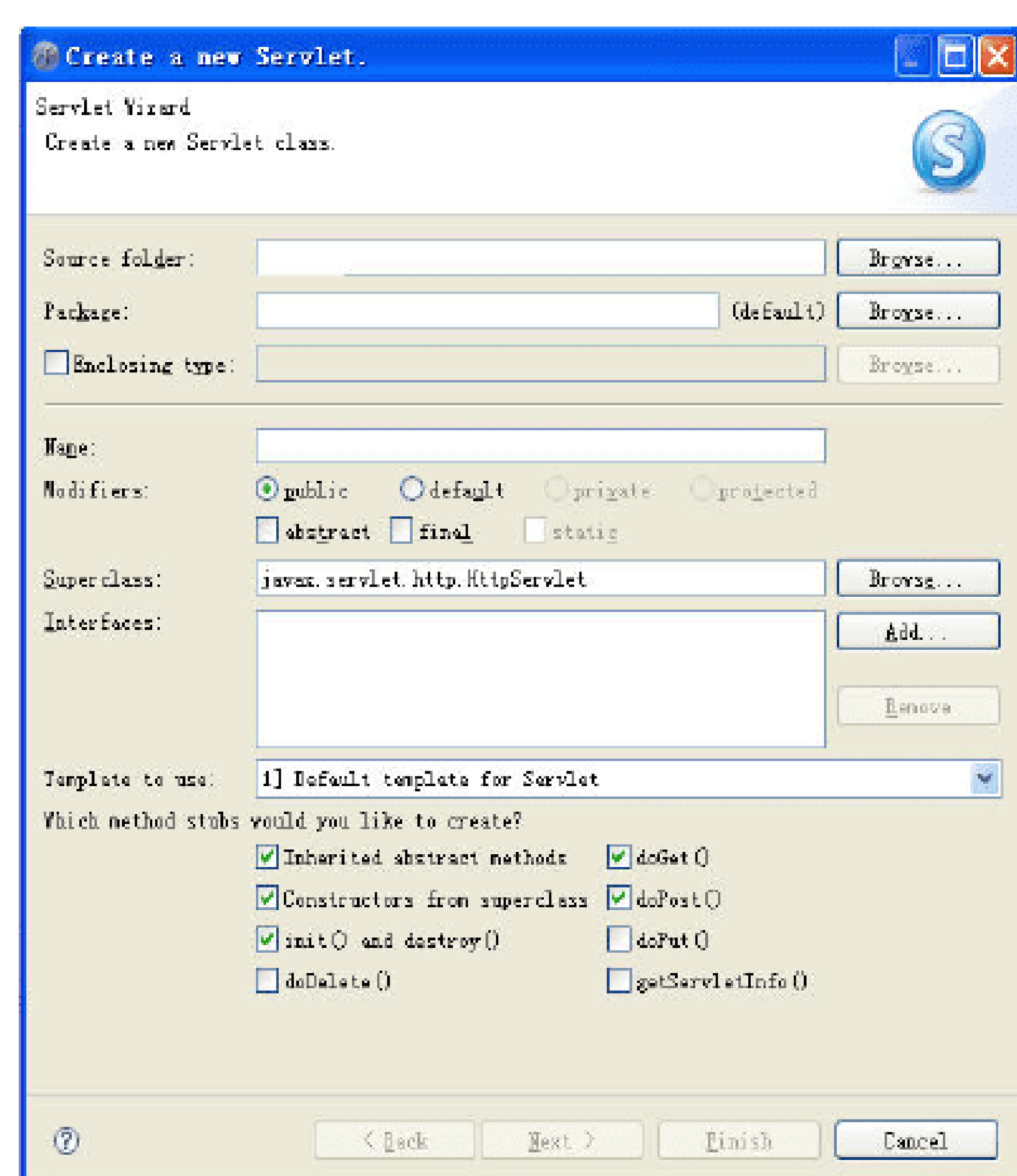


图 2.4　创建 Servlet 程序

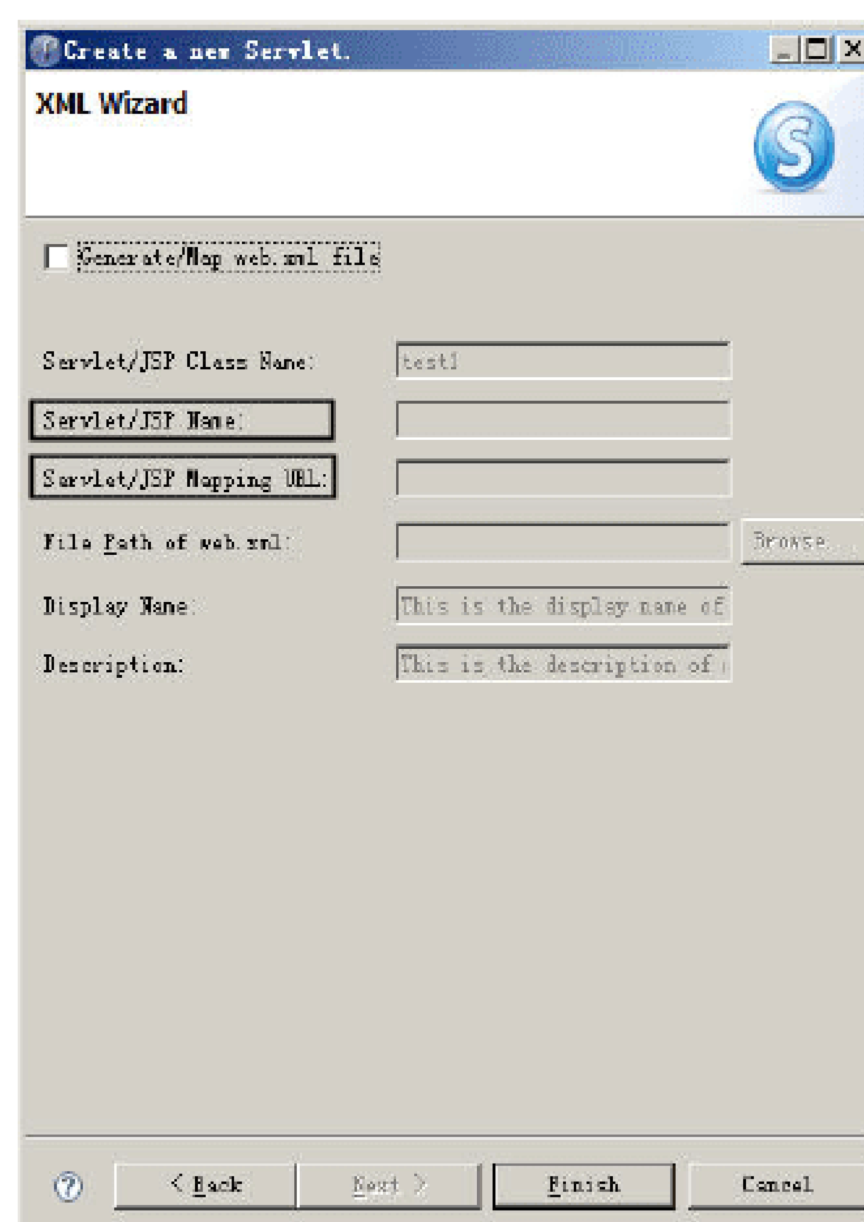


图 2.5　设置 web.xml 文件

经过上述操作，该项目就拥有了 Model 2 模式的各个组件部分。

2.2　Struts 框架的实现

到目前为止，Struts 框架分为两个大的版本体系 Struts 1.x 和 Struts 2.x。虽然 Struts 2.x 提供了与 Struts 1.x 的兼容，但是由于两个版本在体系上存在很大的差异，因此具体开发步骤也不同。

2.2.1　下载和分析 Struts 1.x 框架包

目前 Struts 1.x 最新的版本为 Struts 1.3.10，注意下载时要选择 Full Distribution（完整版本），因为该类型的版本适合于初学者使用。通过访问下载 Struts 1.3.10 的官方网站（http://struts.apache.org/download.cgi#struts1310）来下载该框架，如图 2.6 所示。

各种版本的 Struts 1.3.10 意义如下。

- Full Distribution：Struts 1.3.10 的完整版本。
- Library：Struts 1.3.10 类库。
- Source：Struts 1.3.10 的完整源代码。
- Examples：Struts 1.3.10 的示例应用。
- Documentation：Struts 1.3.10 的相关文档。

下载完 Struts 1.3.10 的完整版后，将下载到的 Zip 文件解压后，该文件夹包含如下的文件结构。

- apps：包含了基于 Struts 1.3.10 的示例应用。
- docs：包含了 Struts 1.3.10 的相关文档，包括 Struts 1.3.10 的快速入门、Struts 1.3.10 的文档和 Struts 1.3.10 API 等内容。
- lib：包含了 Struts 1.3.10 的核心类库。
- src：包含了 Struts 1.3.10 的全部源代码。

目前 Struts 2.x 最新版本为 Struts 2.1.6，同样要选择 Full Distribution（完整版本），通过访问下载 Struts 2.1.6 的官方网站（http://struts.apache.org/download.cgi#struts216）来下载 Struts 2.1.6，如图 2.7 所示。

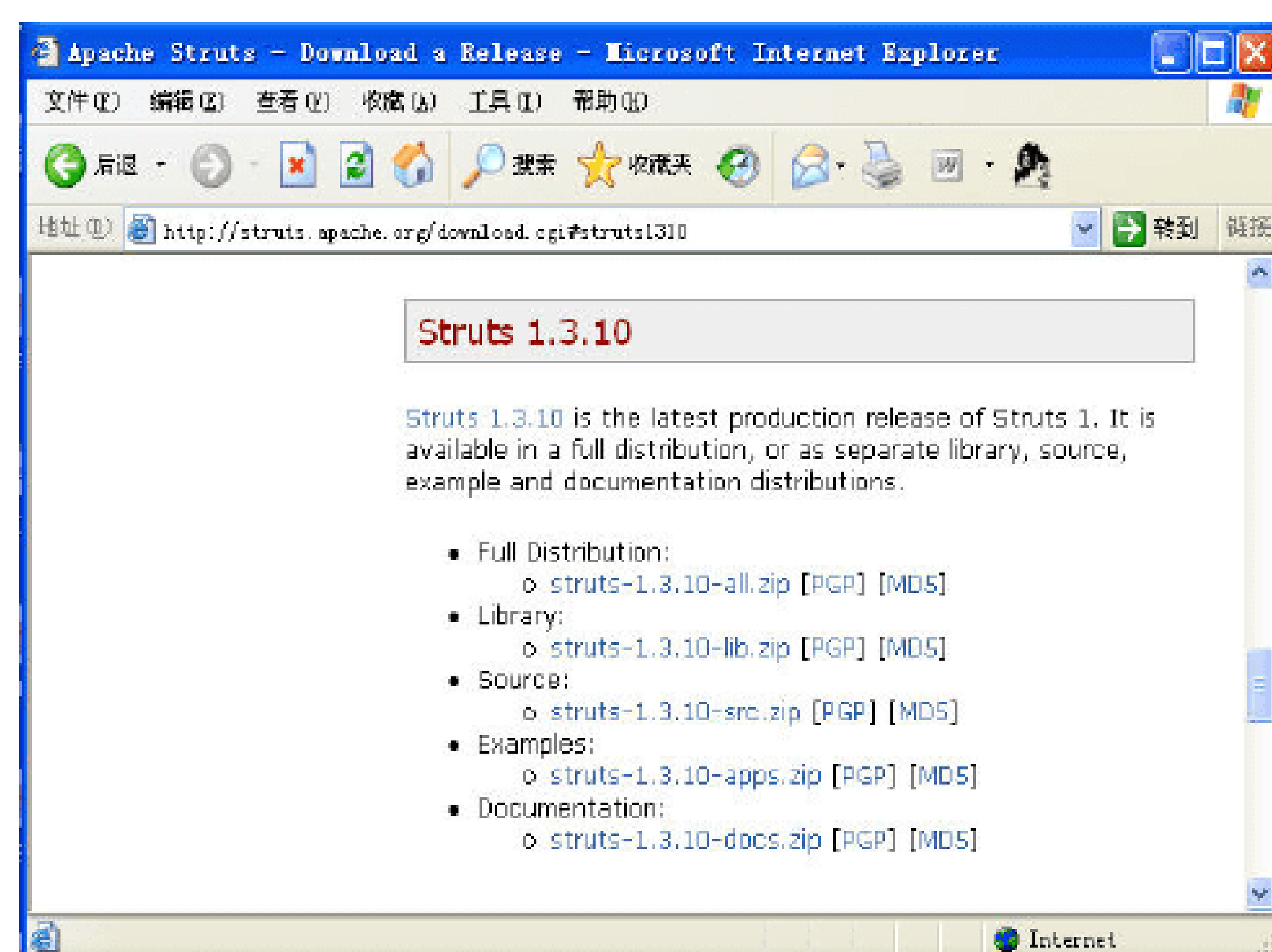


图 2.6　Struts 1.3.10 下载页面

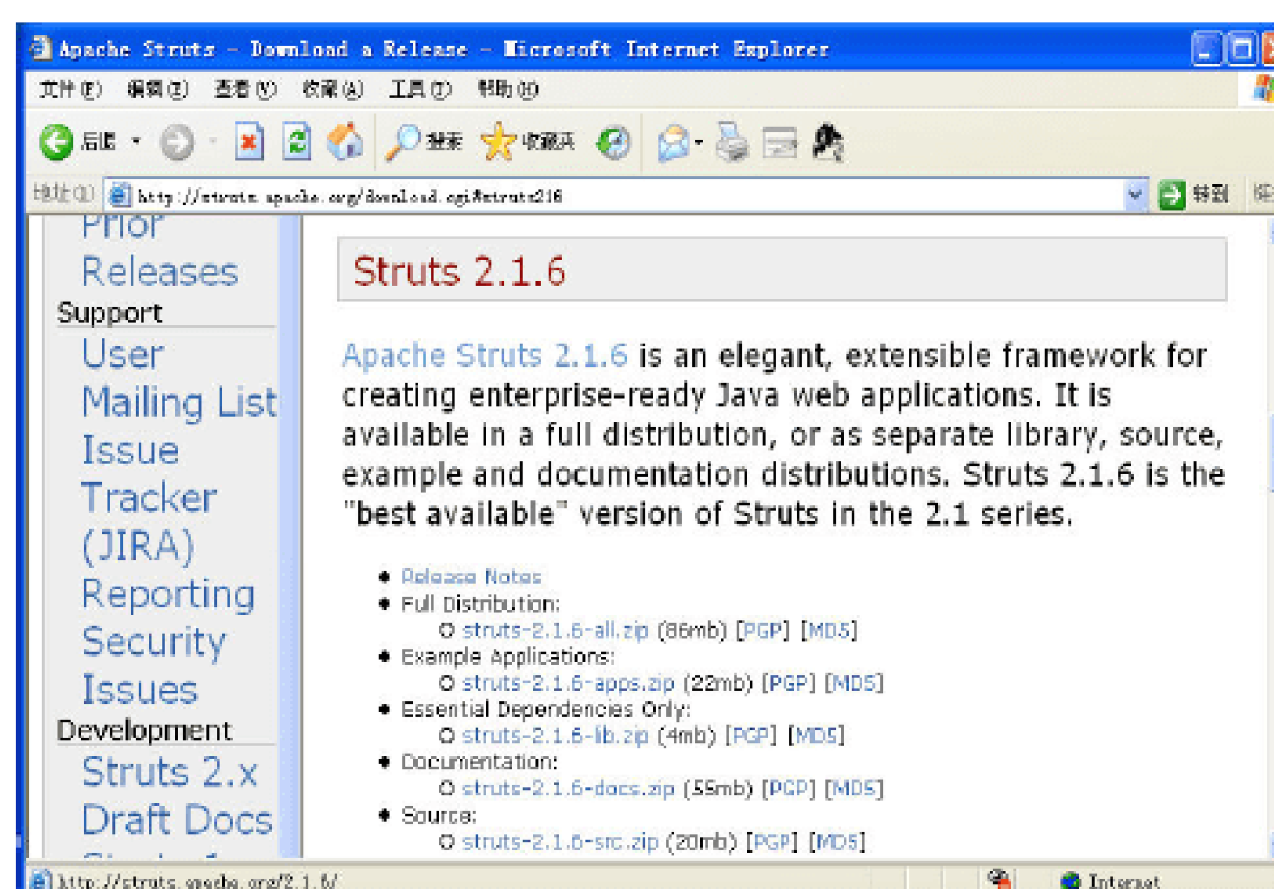


图 2.7　Struts 2.1.6 下载页面

如果想在 Java Web 项目中使用 Struts 2.1.6，只要把 Struts 2.1.6 解压后 lib 文件夹下的 Struts-core-2.1.6.jar、xwork-2.1.2.jar、ognl-2.611.jar、freemarker-2.3.13.ja 和 commons-logging-1.0.4.jar 这 5 个必要的类库，复制到 Java Web 应用的 WEB-INF/lib 路径下就可以。

注意：如果需要在 DOS 窗口下手动编译 Struts 2.1.6 相关的程序，则还需要将 spring-core-2.0.8.jar 添加到系统的 CLASSPATH 环境变量里。

2.2.2　用 MyEclipse 实现 Struts 1.x 框架环境

当利用手工方式来开发 Struts 1.x 框架应用时，虽然能更清楚地理解开发细节，但开发效率却不如使用开发环境的效率高。开发环境 MyEclipse 全方位支持 Struts 1.x 框架，本节将介绍如何在 MyEclipse 开发环境中实现 Struts 1.x 框架环境。

（1）新建一个名为 Struts 1.0 的 Web Project，详细设置如图 2.8 所示。

（2）为项目增加 Struts 1.x 框架的相关类库与文件，需要在 Package Explorer 视图中右击该项目，在弹出的快捷菜单中选择 MyEclipse|Project Capabilities|Add Struts Capabilities 命令（如图 2.9 所示），打开添加 Struts 功能对话框。

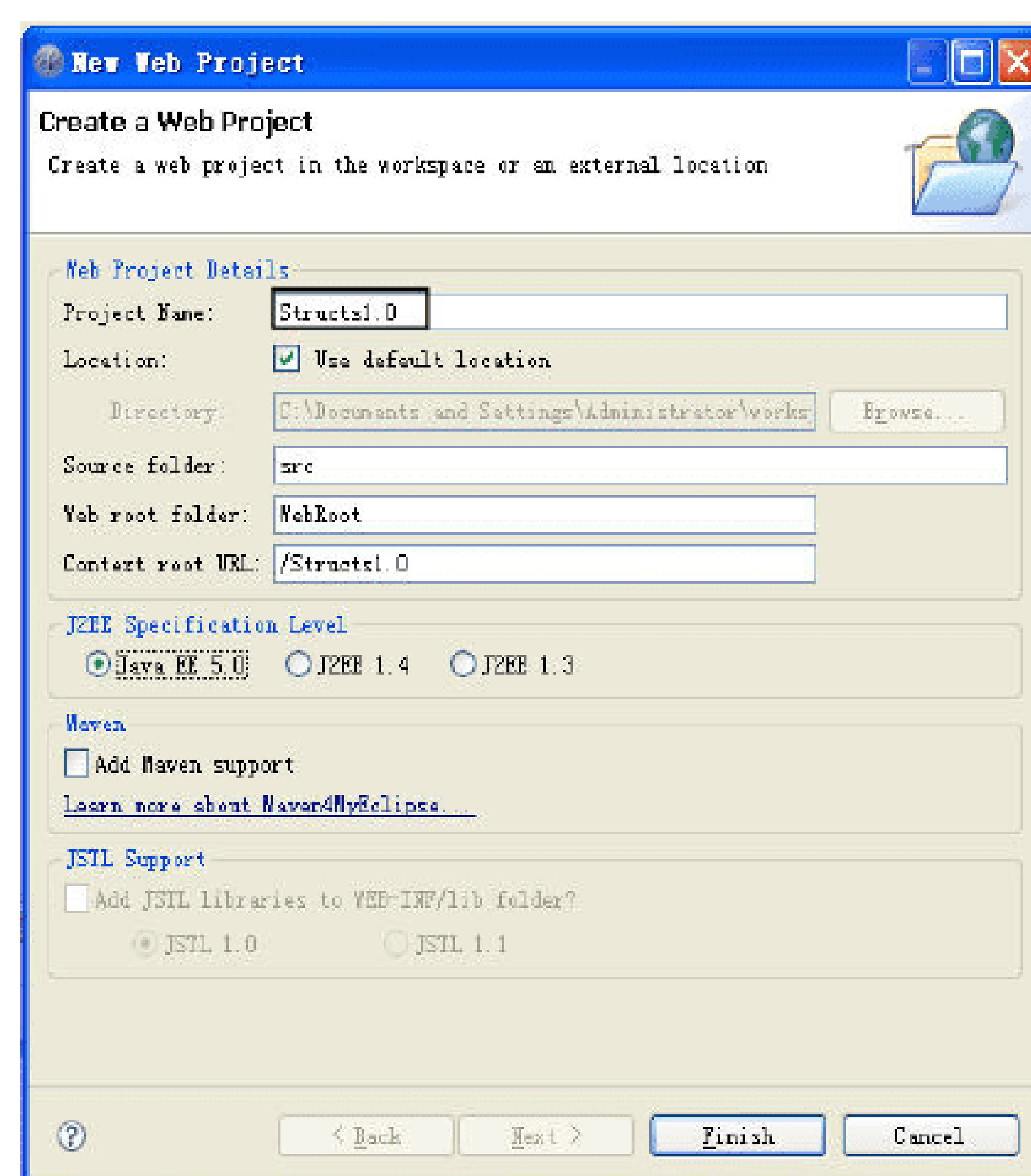


图 2.8　新建 Web Project

注意：该步骤也可以通过选择 MyEclipse|Project Capabilities|Add Struts Capabilities 命令来实现，如图 2.10 所示。

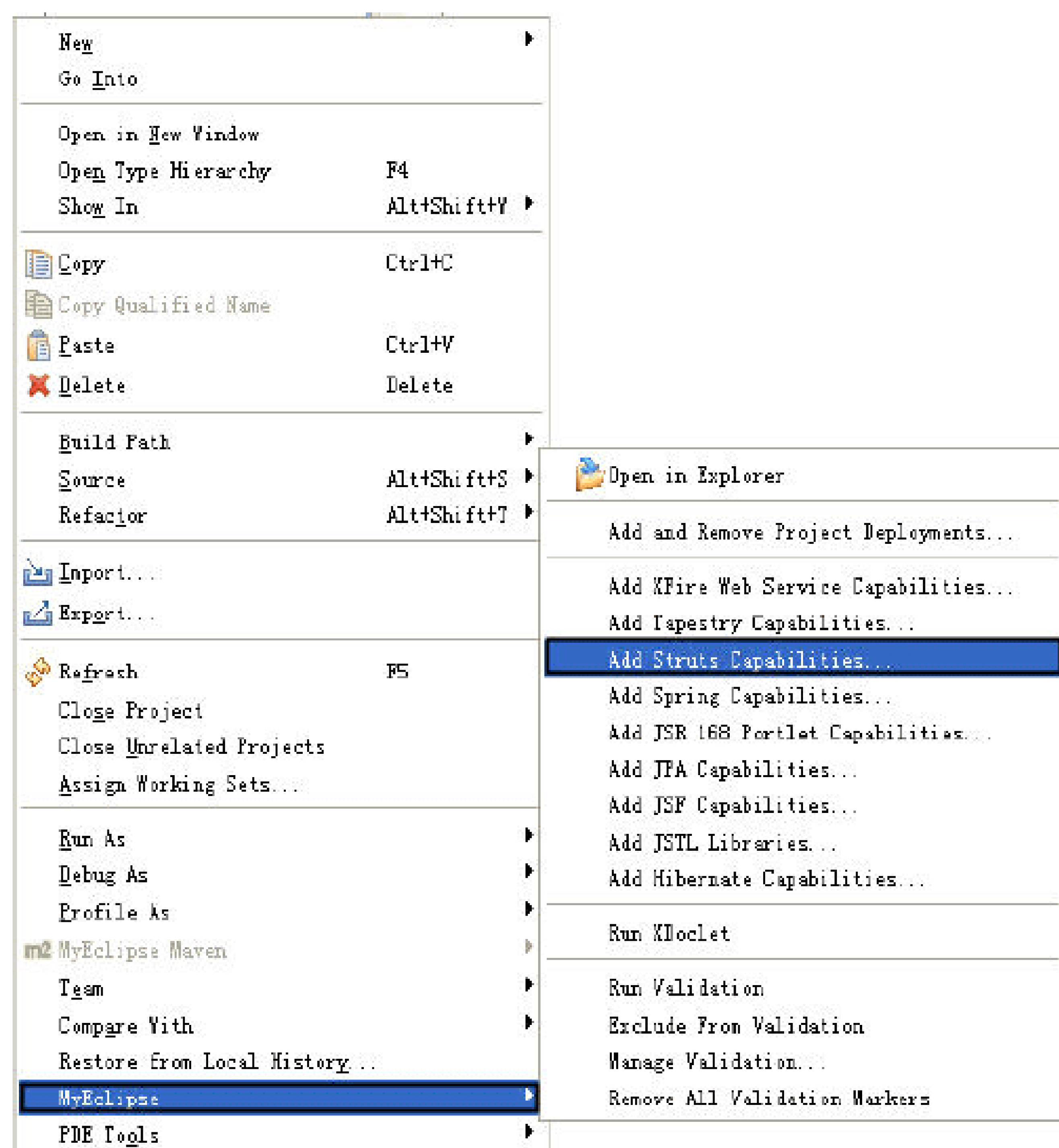


图 2.9　通过右键实现添加 Struts

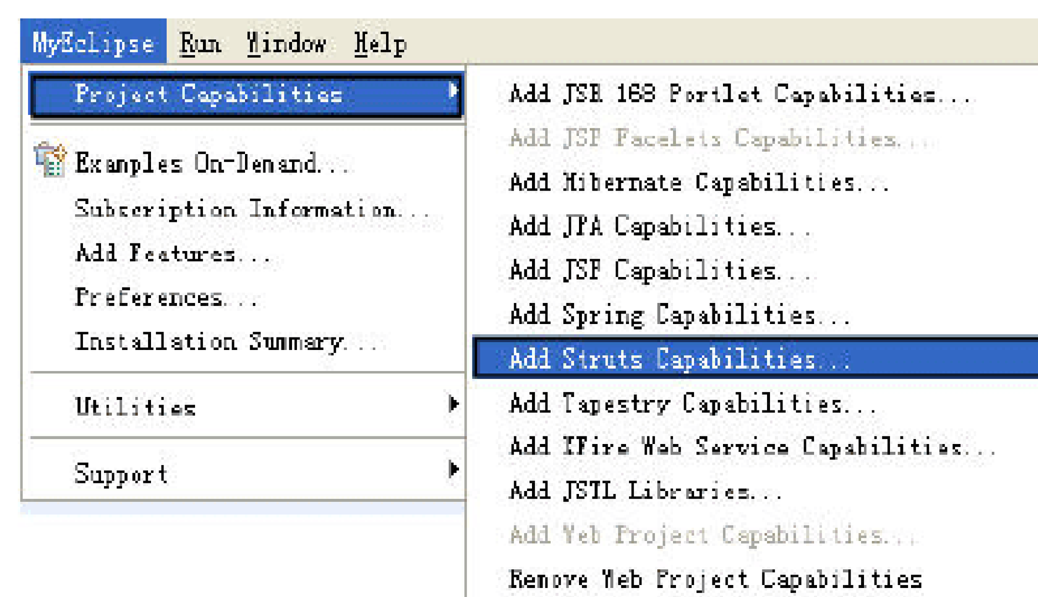


图 2.10　通过菜单实现添加 Struts

在弹出的图 2.11 所示的添加 Struts 功能对话框中，默认值一般来说不需要修改就可以使用，当然也可以根据需要通过修改一些地方来定制将来生成的类。各个选项的具体含义如下。

- ❑ Struts config path：用来设置配置文件（struts-config.xml）的存储位置，该选项一般不需要修改。
- ❑ Struts specification：用来选择 Struts 的版本，设置为 Struts 1.3。
- ❑ ActionServlet name：ActionServlet 的名称，该选项一般不需要修改。
- ❑ URL pattern：指定了将来交给 Struts 控制的 URL 类型，该选项一般选择“*.do”项。
- ❑ Bse package for new classes：用来指定生成类的默认包。该选项一般需要修改，设置为 com.cjg.struts。
- ❑ Default application resources：用来指定资源文件的默认包，该选项一般不需要修改。

当完成增加 Struts 1.x 相关类库与文件操作后，打开 Package Explorer 视图中 Structs 1.0 项目目录，如图 2.12 所示。

- ❑ Struts 1.3 Libraries：为项目添加 Struts 1.3 类库。
- ❑ Struts-config.xml：为项目添加的 Struts 配置文件。

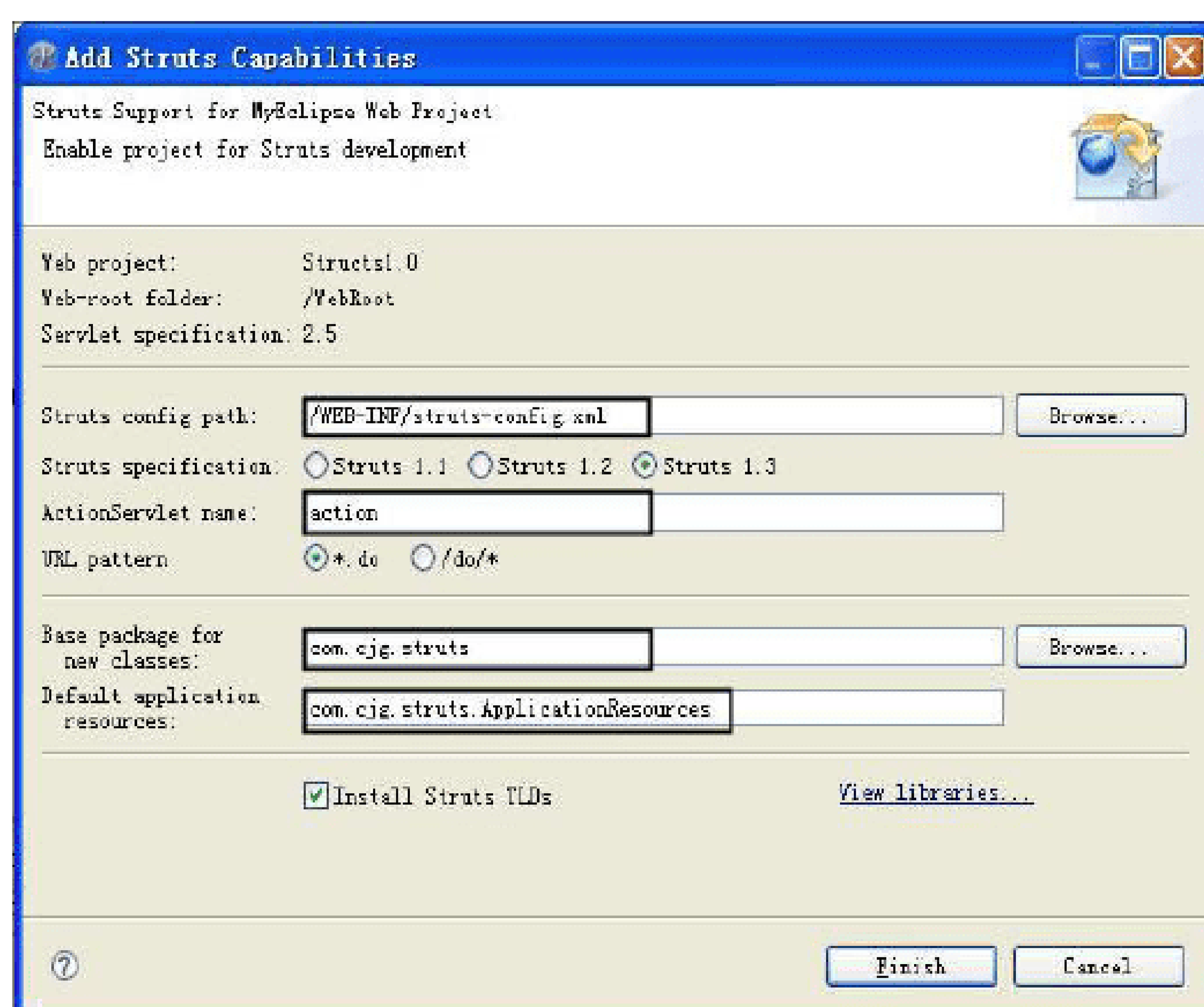


图 2.11　添加 Struts 框架

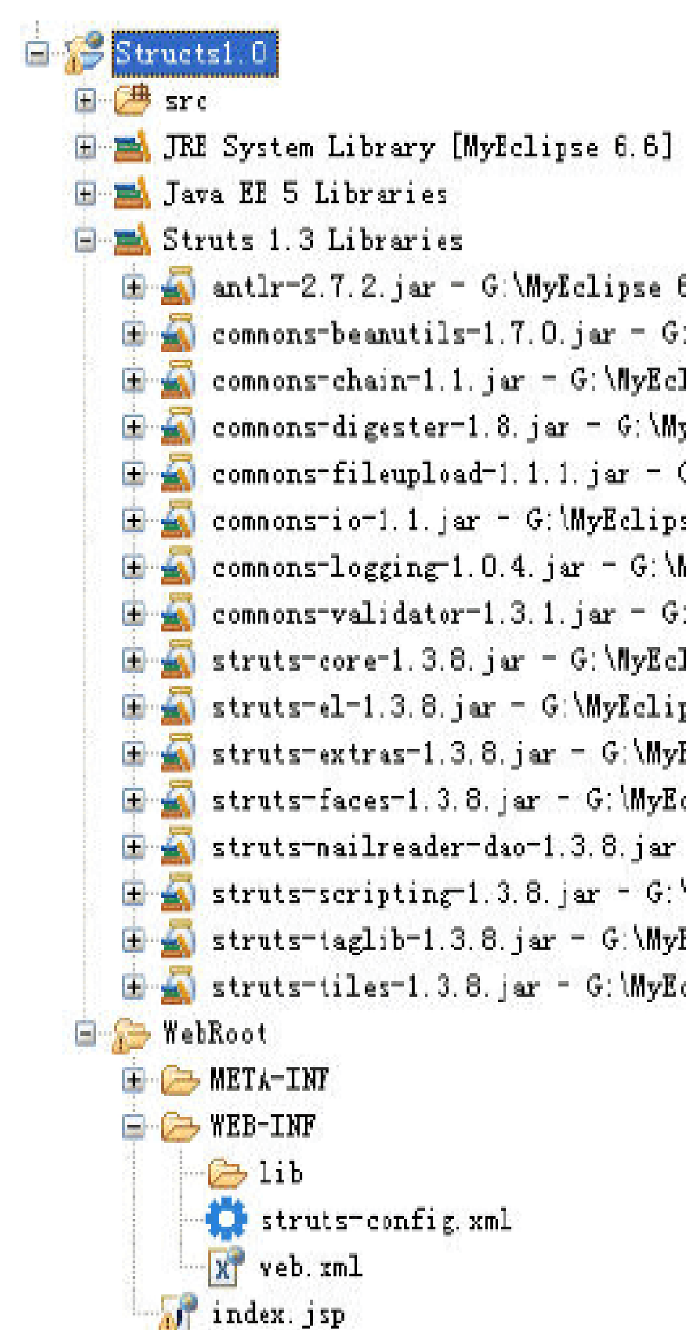


图 2.12　Structs 1.0 项目目录

至此，该项目已经具备了 Struts 1.x 框架的支持。

2.2.3　用 MyEclipse 实现 Struts 1.x 项目

为了便于讲解，本节将在 2.2.2 节完成的开发环境中，利用 Struts 1.x 框架实现一个具体的应用。首先介绍代码的运行背景，用户通过用户名和密码实现登录功能，具体步骤如下。

在介绍之前，先熟悉一下 Struts 配置文件编辑器，通过双击 Struts-config.xml 文件，可以打开 Struts 配置文件编辑器，如图 2.13 所示。

右击画布面板，在弹出的快捷菜单中选择 New 命令就会出现 4 个选项，如图 2.14 所

示。其中选项 Form 表示表单；选项 Action 表示事件；选项 Forward 表示执行完 Action 后所进入的路径，本节将通过该编辑器为 Structs 1.0 项目实现各个组件。

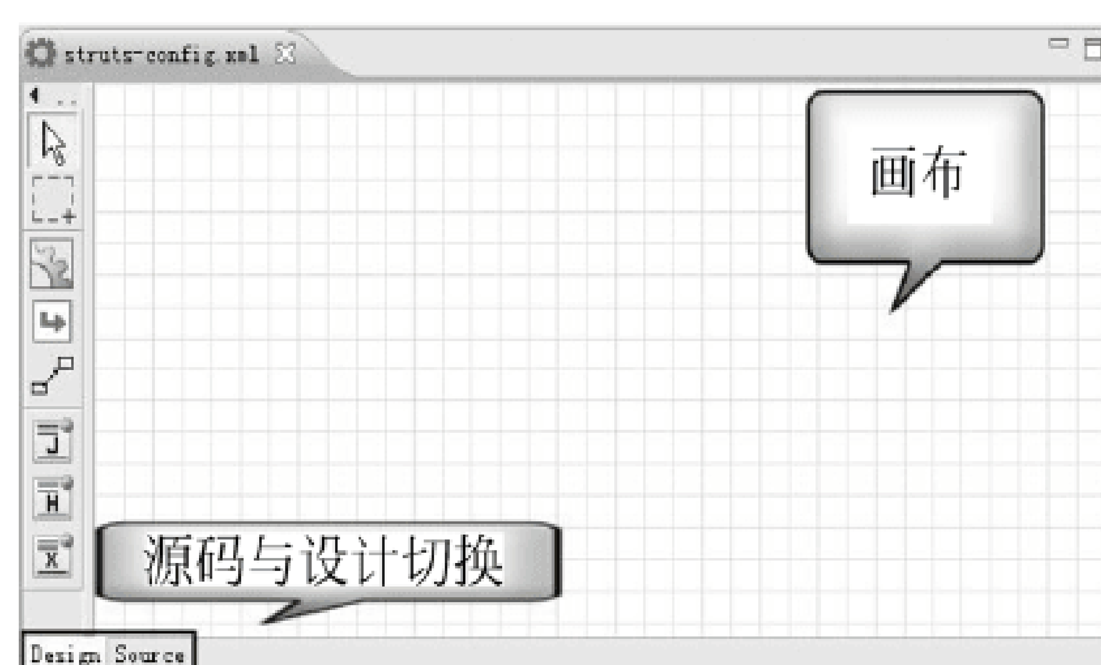


图 2.13　Struts 配置文件编辑器

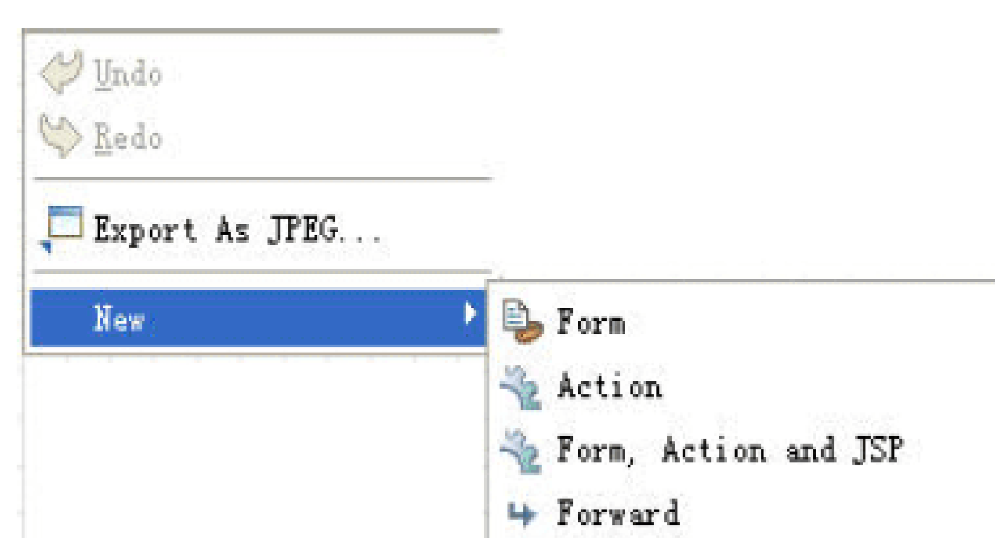


图 2.14　Struts 项目的组件

（1）新建 FormBean，在图 2.14 中选择“Form，Action and JSP”选项，就会弹出如图 2.15 所示的 Create Struts 1.2 FormBean 对话框。

在 Use case 文本框中输入 login，之后 Name 选项就会自动生成用来显示 FormBean 的名字。Form Impl 右侧的单选按钮用来设置 Form 的具体实现类，其中 New FormBean 表示新建 FormBean、Existing FormBean 表示已存在的 FormBean 和 Dynamic FormBean 表示动态的 FormBean。本项目选择 Dynamic FormBean 选项，这样就可以不用创建具体的 ActionForm 来包装页面表单值。

如果想添加 Form 属性，可以通过元素 Form Properties 来实现。在元素 Form Properties 中单击 Add 按钮，弹出 Create Property 对话框，如图 2.16 所示。选项 Name 用来设置属性的名字；Type 用来设置属性的类型；JSP input type 则用来设置显示时的类型。创建的两个属性值如表 2.1 所示。

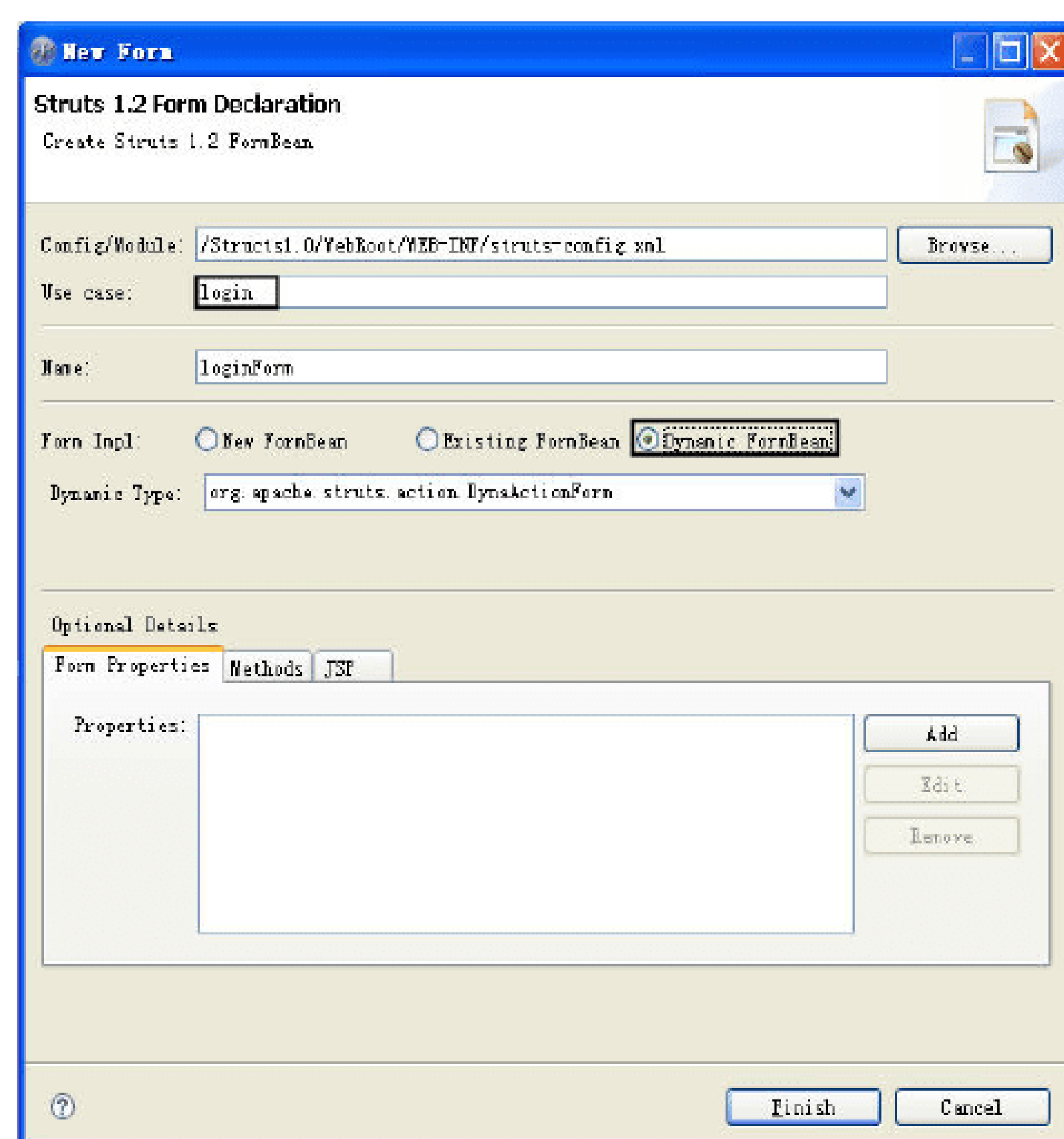


图 2.15　新建 FormBean



图 2.16　创建属性

表 2.1　Form属性的具体值

	Name	Type	JSP input type
用户名属性	username	java.lang.String	Text
密码属性	password	java.lang.String	Password

如果想生成 FormBean 对应的 Struts 表单输入页面，可以通过元素 JSP 来实现，如图 2.17 所示。默认情况下复选框 Create JSP Form 是未选中的，在本项目中先选中该复选框，然后将 New JSP Path 选项值改为“/login.jsp”。

（2）新建 Action，通过单击图 2.14 中的 Action 按钮，就会出现 Create Struts 1.3 Action 对话框（如图 2.18 所示）。在该对话框中大部分的选项值都已经填好了，一般不需要做任何修改。各个选项的具体意义如下。

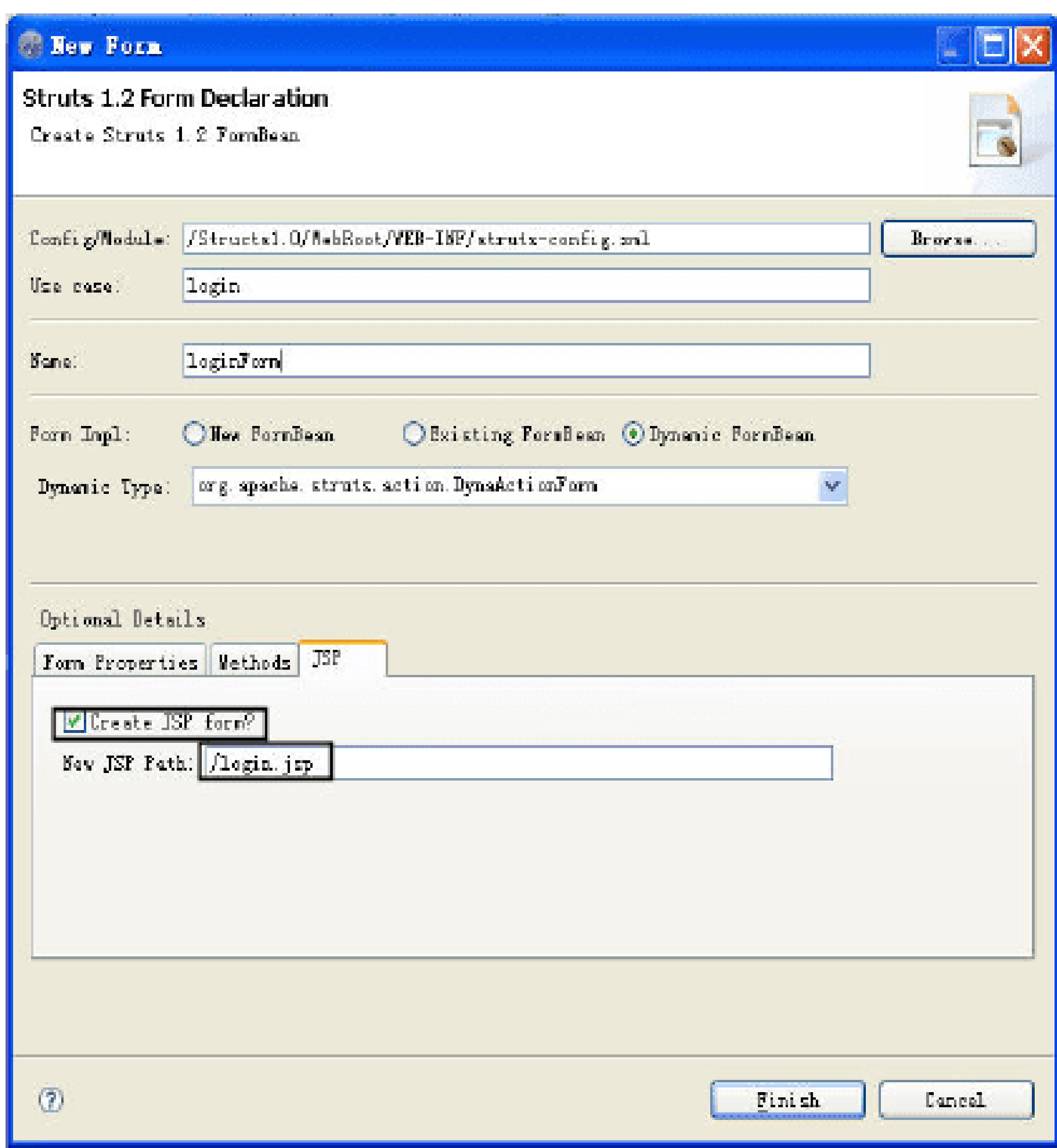


图 2.17　创建 JSP 页面

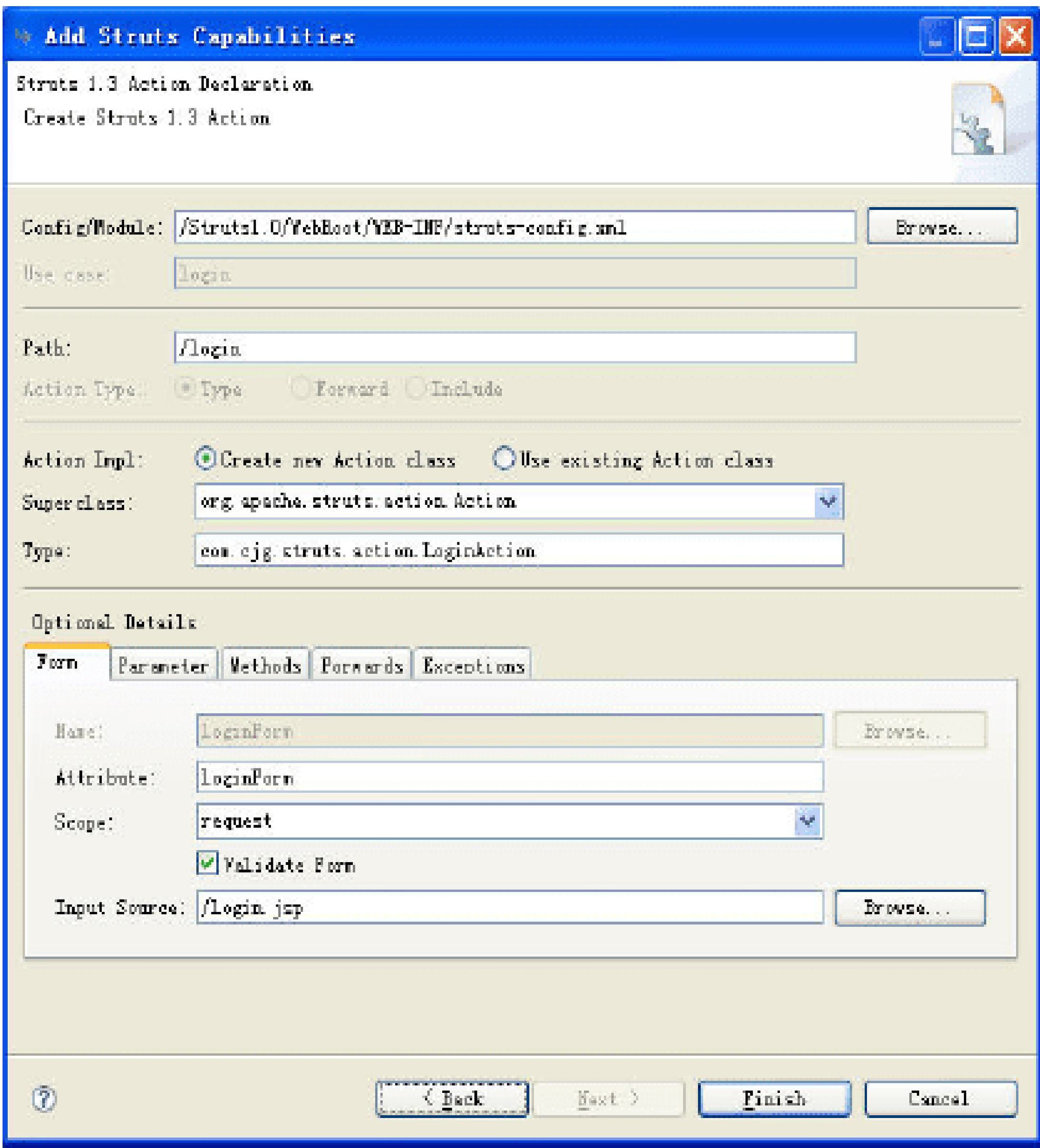


图 2.18　创建 Action

- ❑ Config/Module：指定 Struts 配置文件
- ❑ Path：指定了最后 Action 通过浏览器访问的路径。
- ❑ Action Impl：指定 Action 的类。

（3）新建 Forward，通过图 2.18 中的元素 Forward 来实现，如图 2.19 所示。在元素 Forward 中单击 Add 按钮，弹出 New Forward 对话框，如图 2.20 所示。其中选项 Name 表示转向的名字、Path 用来设置转向的路径。创建的两个转向的相关设置如表 2.2 所示。

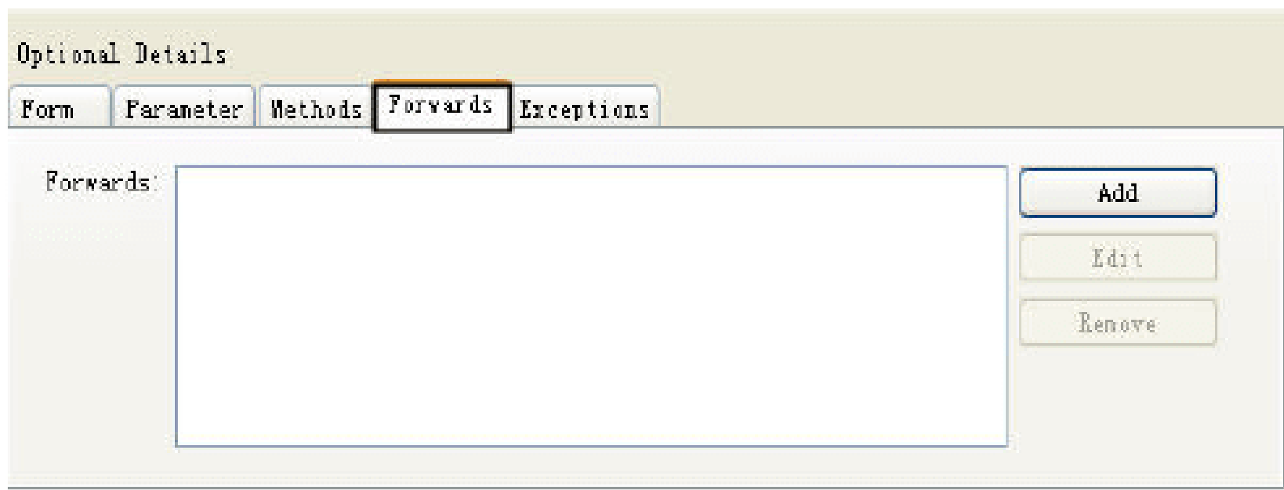


图 2.19　新建 Forward

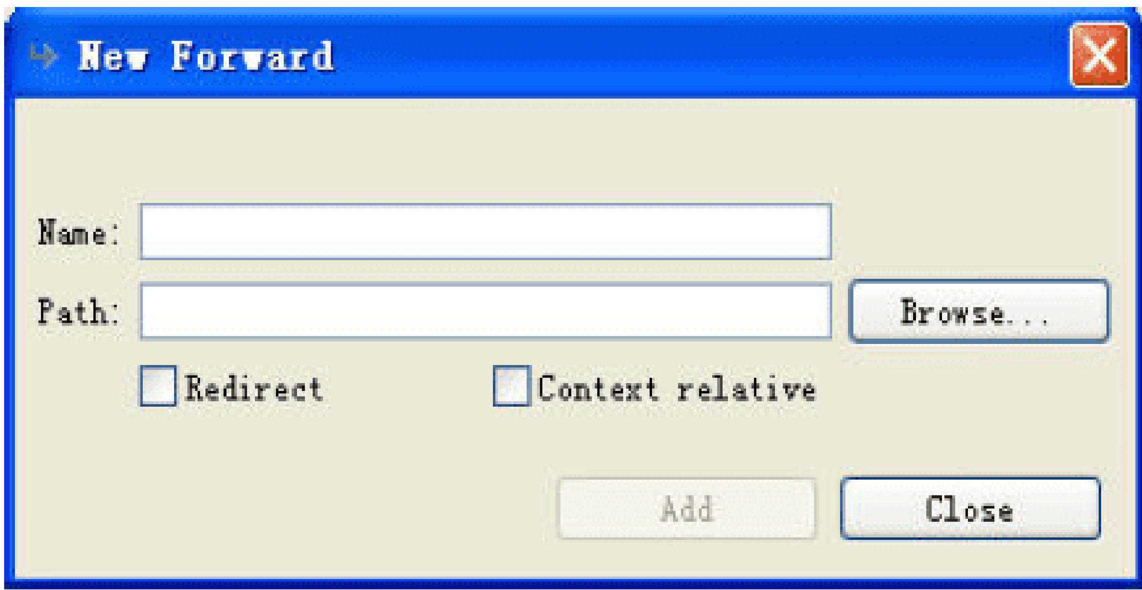


图 2.20　添加 Forward

表 2.2　转向的具体值

	Name	Path
登录成功后的 Forward	success	/success.jsp
登录失败后的 Forward	fail	/fail.jsp

完成上面的配置后，MyEclipse 不仅会自动配置好 struts-config.xml 文件，而且还会用图示的方法显示该项目的流程图，如图 2.21 所示。

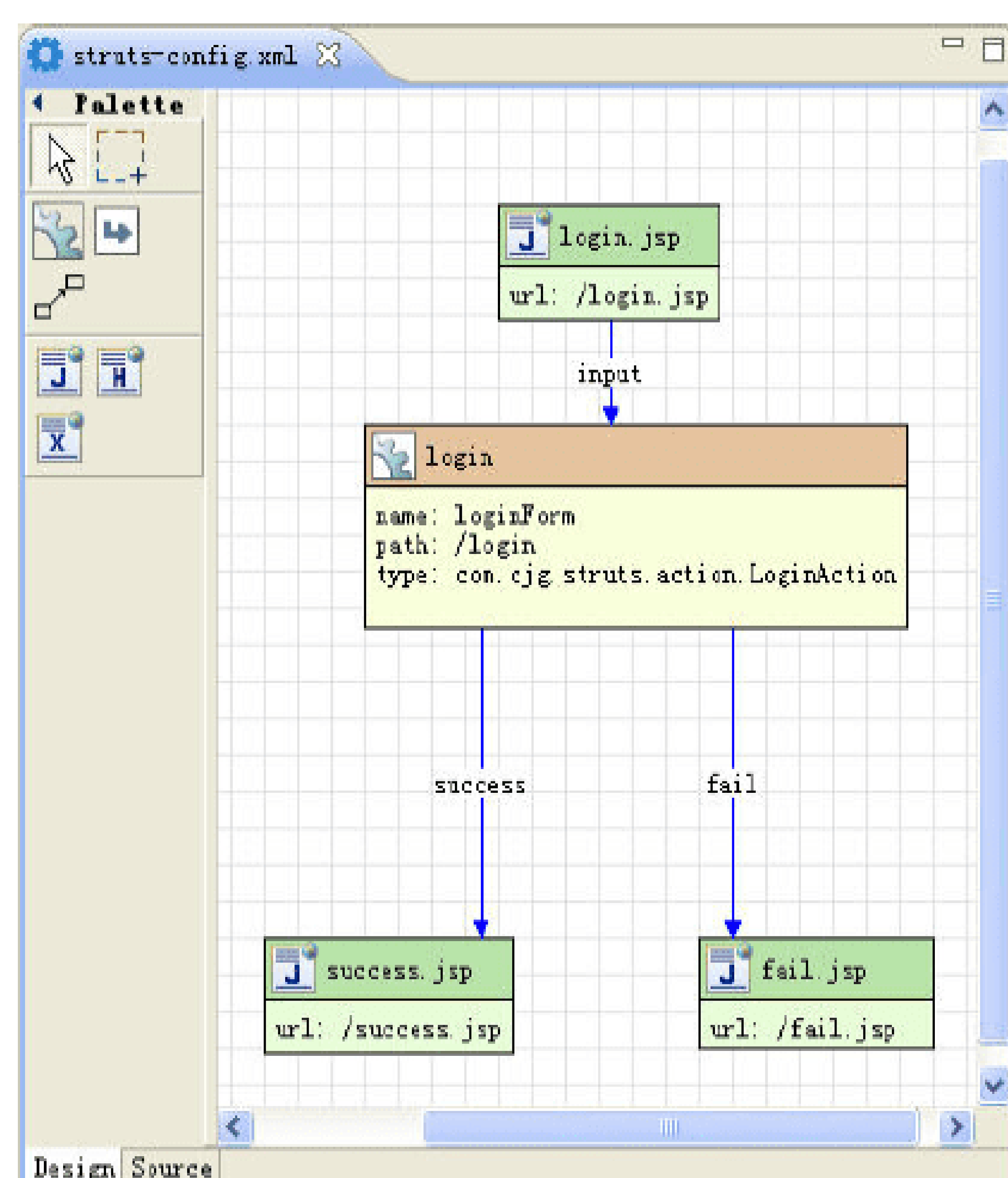


图 2.21　流程图

（4）编辑 Action。LoginAction.java 类作为 Struct 1.x 框架中的 Action 类，用来实现项目的转向流程，具体内容如代码 2.1 所示。

代码 2.1　控制转向：LoginAction.java

```
...
public class LoginAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        DynaActionForm loginForm = (DynaActionForm) form;//获取 ActionForm
        String username = (String) loginForm.get("username");//获取用户名
        String password = (String) loginForm.get("password");//获取密码
        //验证用户名和密码
        if (username.equals("cjg") && password.equals("123456")) {
            return mapping.findForward("success");          //转向成功页面
        }
        return mapping.findForward("fail");                  //转向失败页面
        }
}
```

【代码解析】

在上述代码中，如果 Forward 中封装的用户名和密码分别为 cjg 和 123456 时，则返回 success 字符串，否则就返回 fail 字符串。

（5）编辑 login.jsp、success.jsp 和 fail.jsp 页面。login.jsp 用来实现信息的收集，具体内容如代码 2.2 所示。success.jsp 是登录成功页面，具体内容如代码 2.3 所示。fail.jsp 是登录失败页面，具体内容如代码 2.4 所示。

代码 2.2　信息收集：login.jsp

```
<html:form action="/login">             <!--表单的处理类-->
    <!--用户输入框-->
    用户名 : <html:text property="username"/><html:errors property=
    "username"/>
    <!--密码框-->
```

```
    密  码 : <html:password property="password"/><html:errors property=
    "password"/>
    <html:submit/><html:cancel/>      <!--提交按钮-->
</html:form>
```

代码 2.3 成功后的页面：success.jsp

```
<body>
  你已经登录成功!
</body>
```

代码 2.4 失败后的页面：fail.jsp

```
<body>
   输入的用户名和密码有误，没有通过用户验证。<br>
  <a href="login.jsp"> 请重新输入用户名与密码。</a><br>     <!--链接页面 -->
</body>
```

注意：上述 3 段代码，分别为各自页面的核心部分。

（6）部署、运行项目。通过单击工具栏上的按钮就会出现如图 2.22 所示的部署项目对话框，在该对话框中通过单击右边的 Add 按钮来添加 Web 服务器以完成项目部署。接着单击按钮旁的小三角，在出现的下拉菜单中选择 Tomcat 6.x 下的 Start 菜单项来启动服务器来进行测试，如图 2.23 所示。

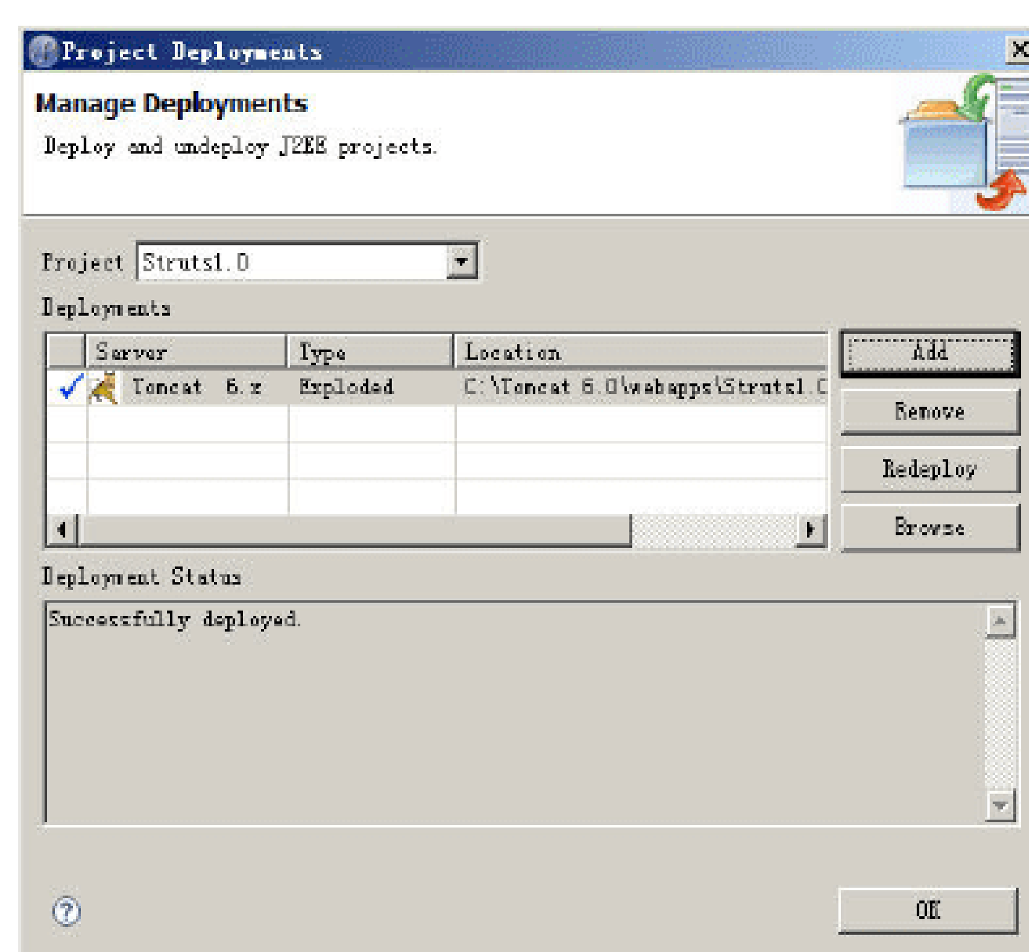


图 2.22 部署项目

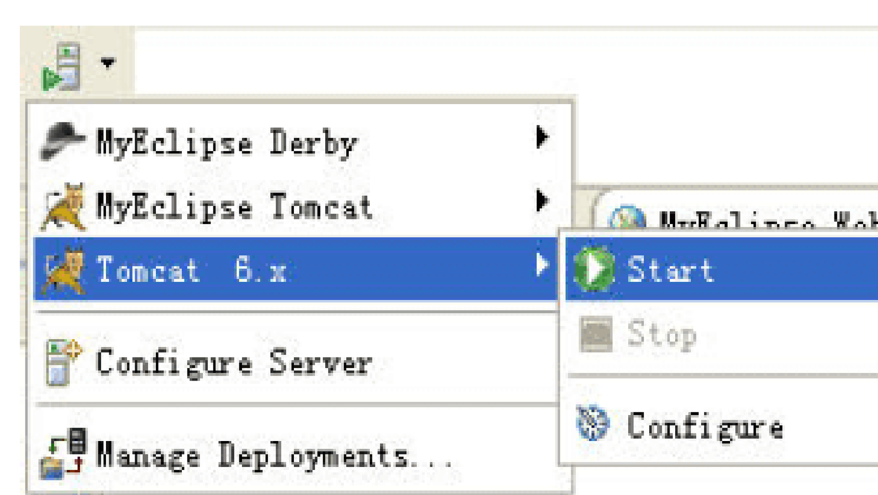


图 2.23 启动服务器

最后打开浏览器，然后输入地址 http://localhost:8080/Struts1.0/login.jsp 来打开登录页面，如图 2.24 所示。如果在登录页面填写的用户名和密码分别为 cjg 和 123456 时，单击 Submit 按钮就会转到如图 2.25 所示的登录成功页面，否则就会转到如图 2.26 所示的登录失败页面。

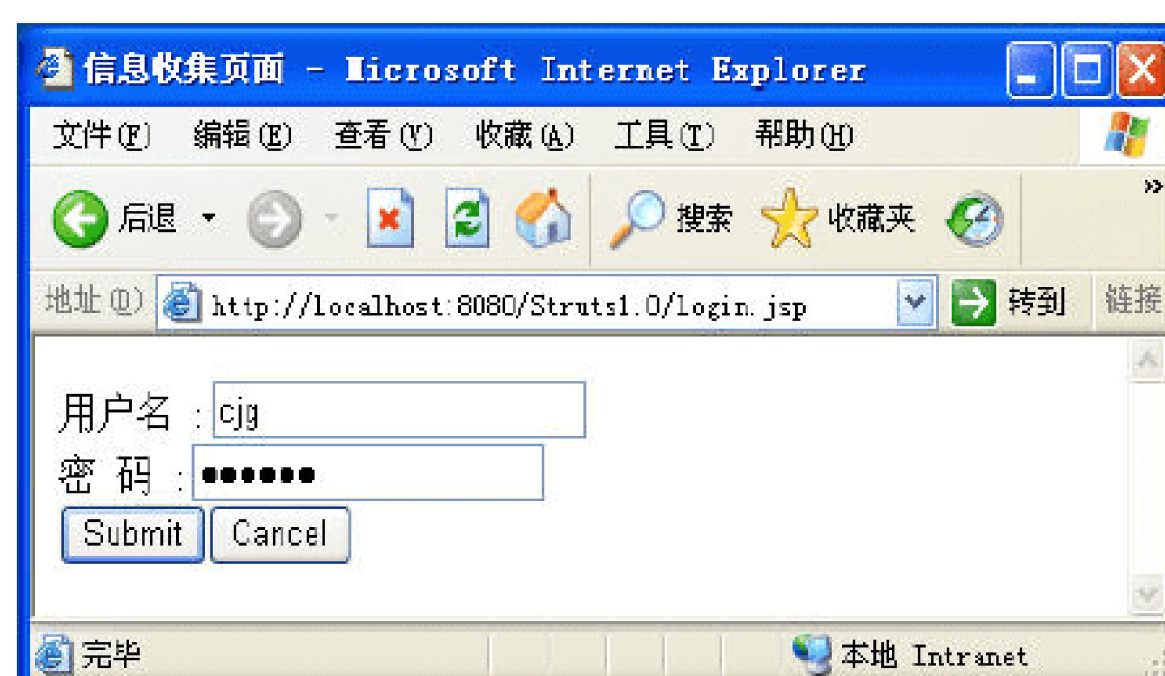


图 2.24 登录页面

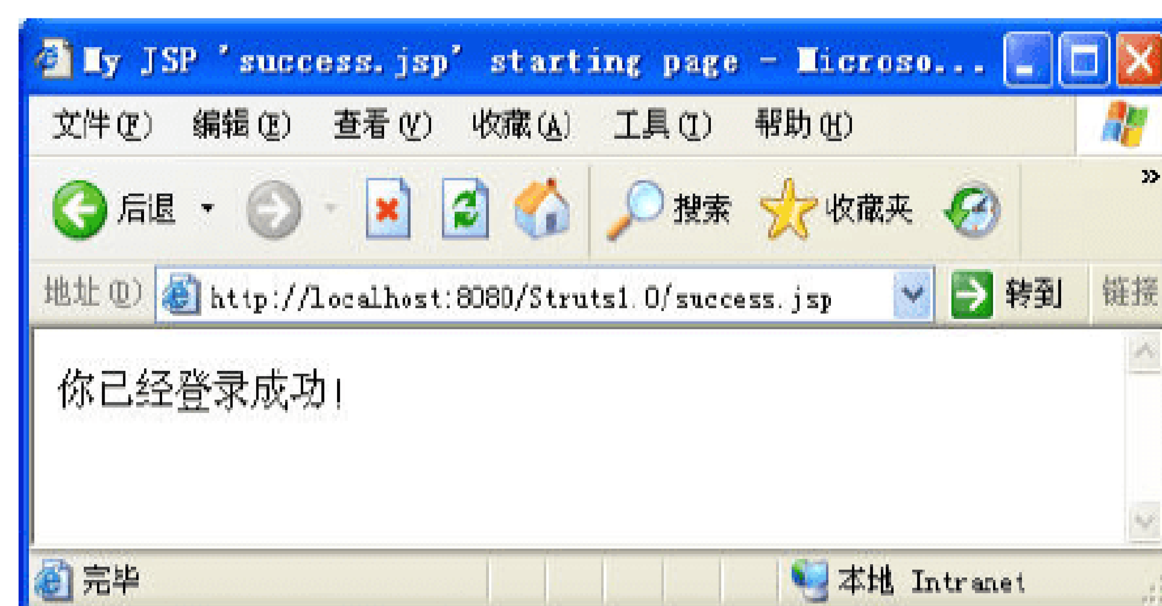


图 2.25 登录成功页面



图 2.26　登录失败页面

2.2.4　分析 Struts 1.x 框架

通过 2.2.3 节介绍的开发步骤基本了解 Struts 1.x 的运行过程，浏览器发出请求，这些请求会被 Struct 1.x 的核心控制器（ActionServlet）拦截，ActionServlet 将请求打包在 ActionForm 中并转交给 Action。该 Action 根据请求处理从相应模型中传过来的封装信息，然后把处理完的信息交给 ActionServlet。最后由 ActionServlet 根据处理后的信息，转向相应的页面，如图 2.27 所示。

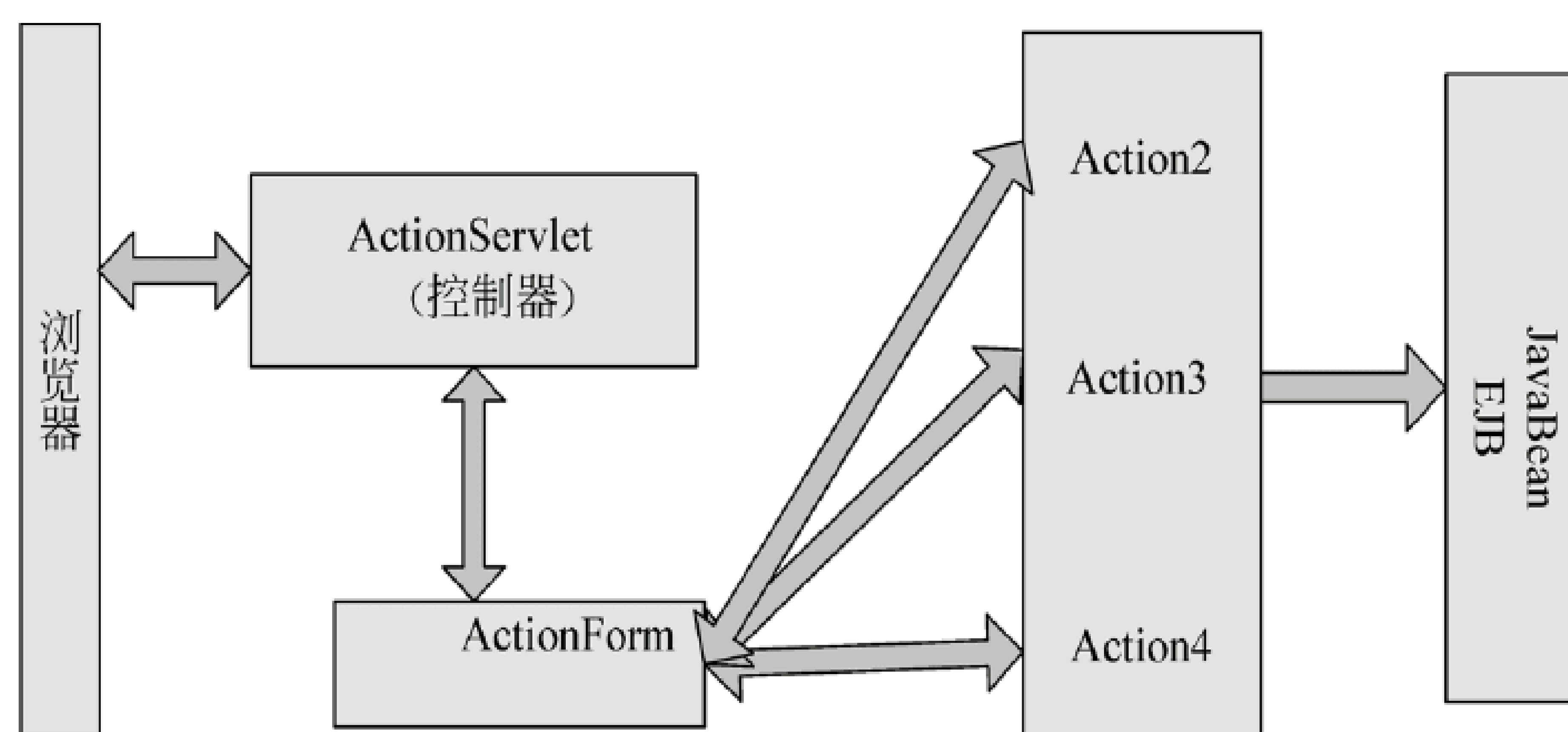


图 2.27　Struts 1.x 框架的流程

通过上述分析可以知道，Struts 1.x 框架由 5 个部分组成：核心控制器 ActionServlet、封装信息的组件 ActionForm、业务控制器 Action、业务逻辑模型和配置文件 struts-config.xml。

- 核心控制器 ActionServlet：该控制器继承于 java.servlet.http.HttpServlet 类，其作用就是截取所有请求，然后根据 struts-config.xml 配置文件把打包到组件 ActionForm 中的请求转发到相应的业务层控制器（Action）上。
- 封装信息的组件 ActionForm：其实际上是一种 JavaBean，主要用于进行视图和控制器之间表单数据的传递。
- 业务控制器 Action：通常每个动作对应一个 Action，其会调用业务逻辑模型的方法、更新业务模型的状态和控制业务流程。
- 业务逻辑模型：对于大型应用来说，其一般由 JavaBean 或 EJB 来实现，主要用于实现业务的逻辑。
- struts-config.xml 文件：ActionServlet 会根据该文件决定将请求发给哪个 Action 对象。

注意：业务流程由 Action 类来实现，而业务逻辑由模型来实现。

在知道了 Struts 1.x 框架的组成部分后，下面来讲解 Struts 1.x 框架的工作流程，当 ActionServlet 接受到一个请求时，将执行如下流程。

（1）把请求信息保存到 ActionForm 对象中，根据配置信息决定是否需要表单验证，如果需要，则调用 ActionForm 对象的 vaildate()方法。

（2）ActionServlet 根据配置文件信息决定把请求转发给哪个 Action，如果相应的 Action 实例不存在就创建该实例，然后调用该 Action 的 execute()方法。

（3）Action 的 execute()方法会返回一个 ActionForward 对象，ActionServlet 会把请求转发给 ActionForward 对象指向的 JSP 组件。

（4）ActionForward 对象指向的 JSP 组件生成动态网页返回给浏览器。

2.2.5　用 MyEclipse 实现 Struts 2.x 框架环境

为了与现实职场工作方式相一致，本节将利用 MyEclipse 开发环境以动态站点的方式来实现 Struts 2.x 框架环境。为什么要使用动态站点方式开发呢？主要因为这种方式不是以文件方式来组织文件目录，而是通过写入的组件来组织文件目录。具体步骤如下。

（1）新建一个名 Structs2 的 Web Project，通过选择 File|New|Other 命令打开 New 对话框（如图 2.28 所示），在该对话框中选择 Dynamic Web Project 项，然后单击 Next 按钮，就会出现如图 2.29 所示的 New Dynamic Web Project 对话框，在该对话框中进行详细的设计后单击 Finish 按钮完成项目的创建。

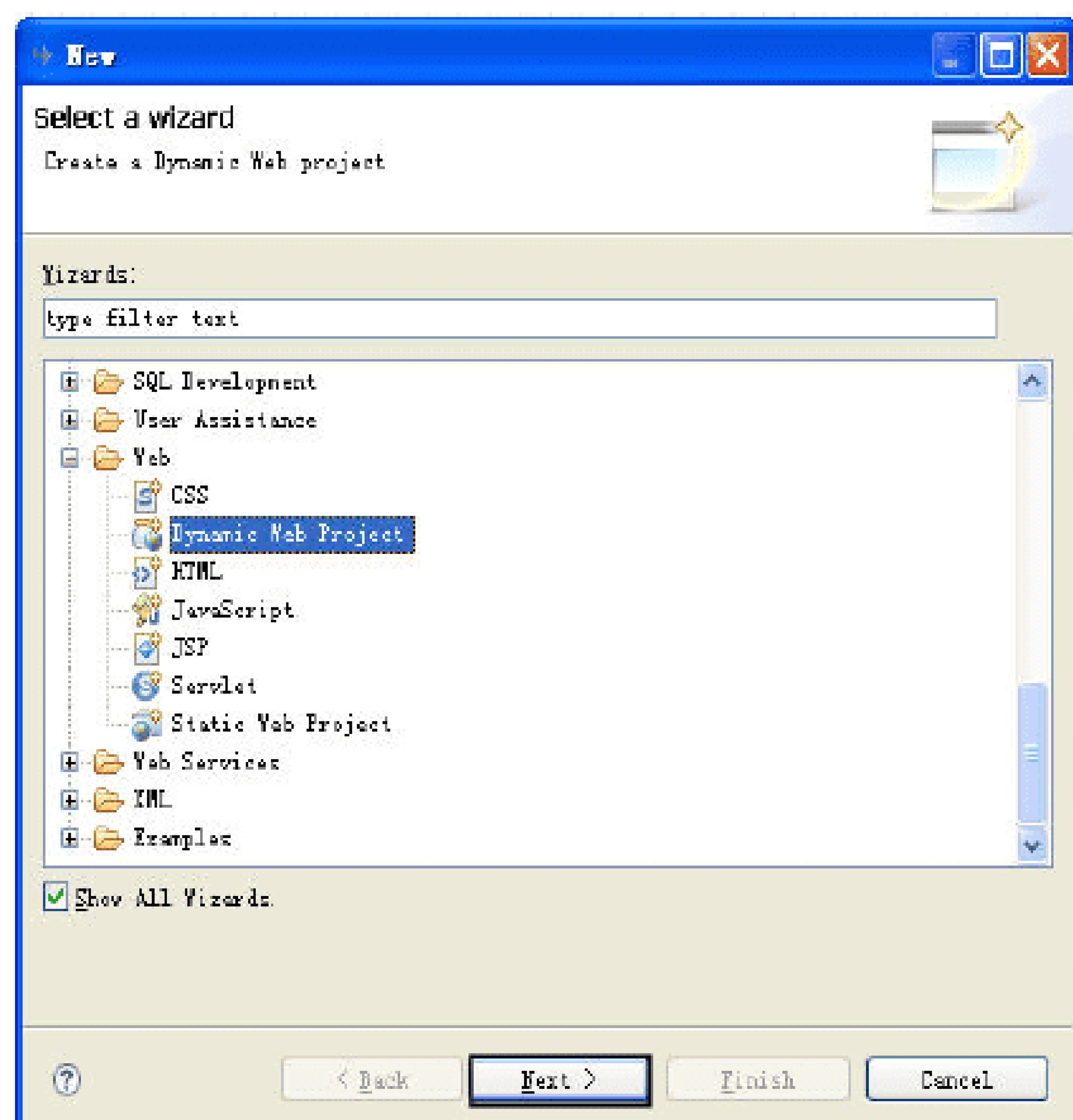


图 2.28　新建对话框



图 2.29　创建动态项目

当使用动态站点方式开发项目时，在 Project Explorer 视图中要以 WTP Java EE 方式显示 Structs2 项目，该项目的目录如图 2.30 所示。

- Java Resource:src：项目的源文件。
- build：该目录可以实现打包处理。
- WebContent：项目的根目录。

🔔注意：在使用动态站点方式开发项目时，最好使用“WTP Java EE”方式来显示目录，因为其是以功能方式来分类。

如何利用 WTP Java EE 方式显示项目时呢？可以通过单击工具栏上的按钮来打开如图 2.31 所示视图对话框，在该对话框中选择 WTP Java EE 项就可以。

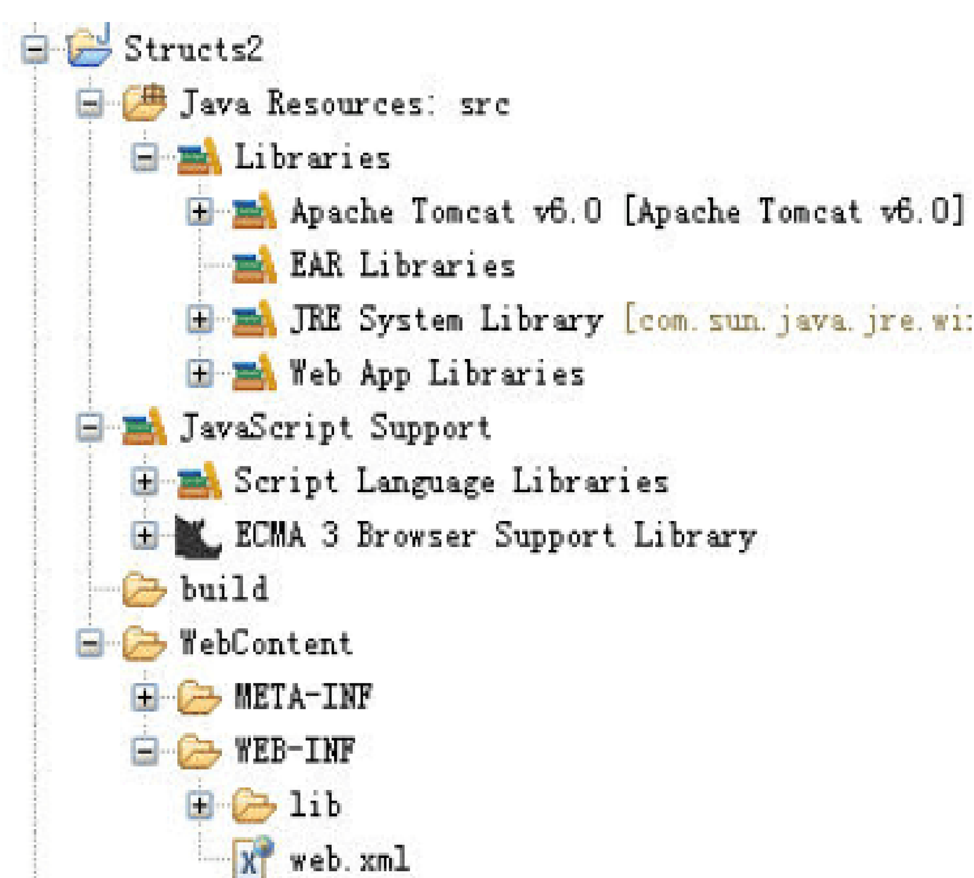


图 2.30　Structs2 项目目录

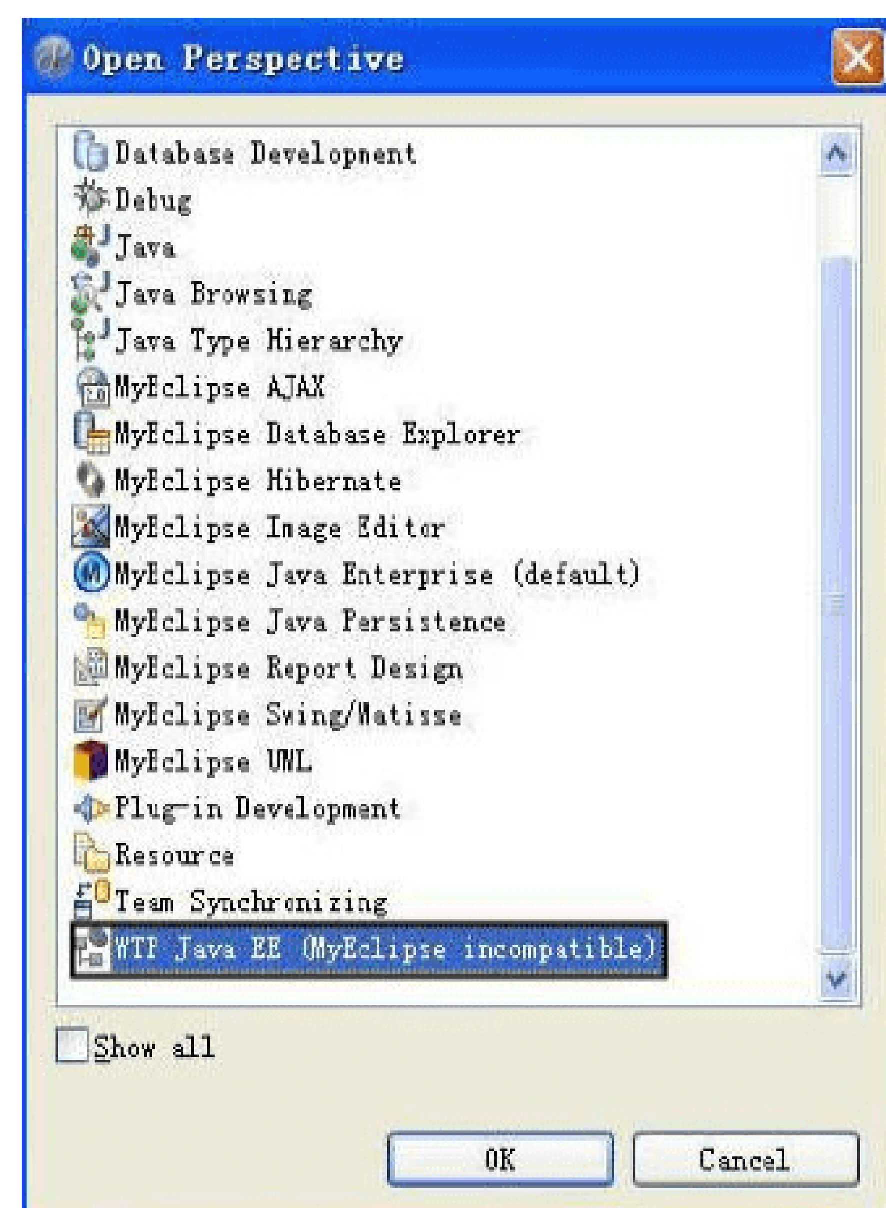


图 2.31　打开视图

（2）为项目增加 Struts 2.x 相关类库与文件，让项目支持 Struts 2.x 框架。这个操作可以先把 Struts 2.x 框架的核心类库，即将 Struts 2.1.6 框架下 lib 路径下的 struts2-core-2.1.6.jar、xwork-2.1.2.jar、oro-2.0.8.jar、freemarker-2.3.13.jar 和 commons-logging-l.0.4.jar 必要包，增加到 Package Explorer 视图的 Structs2.0/Web Root/WEB-INF/lib 的目录下，展开导航树后如图 2.32 所示。

然后修改 web.xml 文件，定义 Struts 2.x 框架的核心 Filter，代码 2.5 使该项目具备了 Struts 2.x 框架的支持。

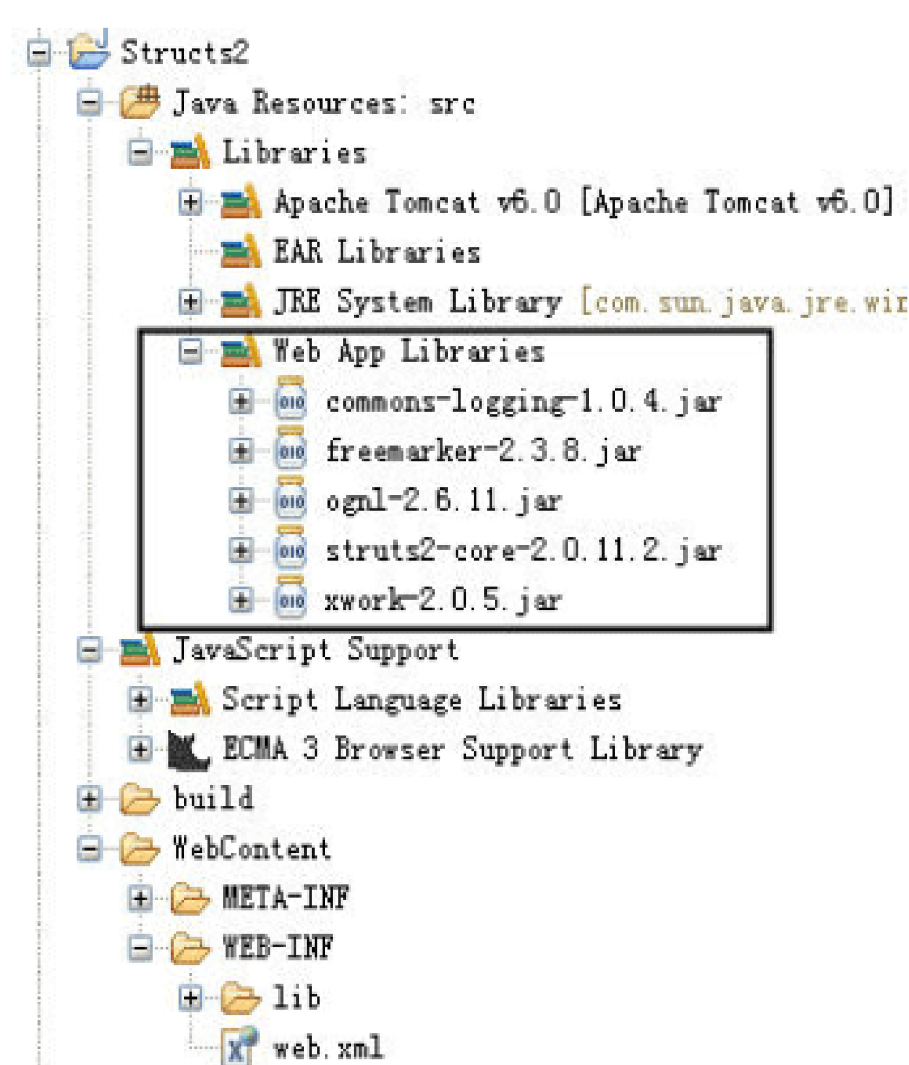


图 2.32　Struts2 项目目录

代码 2.5　定义 Filter：web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns=
"http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
  <filter>                    <!--使用核心包的转发包中的过滤转发来设置过滤器-->
    <filter-name|struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
  </filter>
```

```
  <filter-mapping>           <!--过滤所有的请求-->
    <filter-name|struts2</filter-name>
    <url-pattern|/*</url-pattern>
  </filter-mapping>
</web-app|
```

至此，该项目已经具备了 Struts 2.x 框架的支持。

2.2.6　用 MyEclipse 实现 Struts 2.x 项目

为了便于讲解，本节将在 2.2.5 节完成的开发环境中，利用 Struts 2.x 框架实现一个具体的应用。首先介绍代码的运行背景，就是把项目 Struts1.0 的功能用 Struts 2.x 框架来实现，具体步骤如下。

（1）新建 Action。在 Project Explorer 视图中右击 Structs2 下的 Java Resources:src 目录，在弹出的快捷菜单中选择 New|Package 选项，弹出创建包的对话框。用该向导创建一个名为 com.cjg.struts 的包，然后在该包中新建一个名为 Login 的 Java 文件，代码 2.6 是实现了业务逻辑组件代码。

代码 2.6　逻辑组件：Login.java

```
...
public class Login extends ActionSupport {
    private String username;                    //定义了用户名属性
    private String password;                    //定义了密码属性
    //省略相关属性的 get()和 set()方法
...
    public String execute() throws Exception {
        if ("cjg".equals(this.getUsername().trim())//测试获取的用户名和密码
                && "123456".equals(this.getPassword().trim())) {
            return "success";                        //转向成功页面
        } else {
            this.addFieldError("username", "用户名或密码错误，请重新输入。");
            this.setUsername("");
            this.setPassword("");
            return "failer";
        }
    }
    public void validate() {                         //检验用户名和密码
        if ((null == this.getUsername())
                || ("".equals(this.getUsername().trim()))) {
            this.addFieldError("username", "用户名不能为空。");
        }
        if ((null == this.getPassword())
                || ("".equals(this.getPassword().trim()))) {
            this.addFieldError("password", "密码不能为空。");
        }
    }
}
```

（2）配置 struts.xml 文件。在 Project Explorer 视图中右击 Java Resources:src 目录，然后在弹出的快捷菜单中选择 New|Other|File 命令，弹出创建文件的对话框。通过该对话框创建一个名为 struts.xml 的配置文件，Struts2 框架应用中的 Action 都被配置在 struts.xml 文

件中，代码 2.7 用来实现 Action 的配置。

代码 2.7　Action 配置：struts.xml

```
...
<struts>
<!-- 使用默认的 struts 的配置文件 -->
    <include file="struts-default.xml" />
    <!-- 创建自己的包，该包必须继承默认包-->
    <package name="struts2" extends="struts-default">
        <!-- 配置 longin 请求 Action-->
        <action name="login" class="com.cjg.struts.Login">
            <result name="input"|/login.jsp</result>
                                <!--当标记为 inpupt，转向的页面 -->
            <result name="success"|/result.jsp</result>
                                <!--当标记为 success，转向的页面 -->
            <result name="failer"|/login.jsp</result>
                                <!--当标记为 failer，转向的页面 -->
        </action>
    </package>
</struts>
```

【代码解析】

在上述代码中，元素<action>中属性 name 表示请求的名称，class 表示处理该请求的具体执行类。

（3）创建 login.jsp 和 success.jsp 页面。login.jsp 页面用来实现信息的收集，代码 2.8 是 login.jsp 页面的核心内容，而代码 2.9 则为登录成功后的页面的核心内容。

代码 2.8　信息收集页面：login.jsp

```
...
<body>
<s:form action="login">                                 <!--表单的处理类-->
        <s:textfield name="username" label="用户名："></s:textfield>
                                                        <!--用户输入框-->
        <s:password name="password" label="密码："></s:password>
                                                        <!--密码框-->
        <s:submit ></s:submit>                          <!--提交按钮-->
</s:form>
</body>
...
```

【代码解析】

在上述代码中，元素<s:form>的 action 属性值必须为 struts.xml 配置过的请求名，即 login。

代码 2.9　登录成功后的页面：result.jsp

```
...
  <body>
    username: ${requestScope.username }<br>        <!--输出用户名 -->
    password: ${requestScope.password }            <!--输出密码 -->
  </body>
...
```

（4）运行项目。首先要把该项目导出到 Tomcat 的 webapps 目录下。可以通过菜单 File|Export 打开 Export（导出）对话框，如图 2.33 所示。然后在 Destination 选项的右侧单击 Browse 按钮，选择 Tomcat 的根目录\webapps 目录。最后单击 Finish 按钮就可以把该项目导出到 Tomcat 的目录。

启动 Tomcat 服务器后，打开浏览器，然后输入地址 http://localhost:8080/Structs2/login.jsp 打开登录页面，如图 2.34 所示。在登录页面填写的用户名和密码分别为 cjg 和 123456，单击 Submit 按钮就会转到如图 2.35 所示的登录成功页面，否则就会转到如图 2.36 所示的登录失败页面。



图 2.33　导出项目

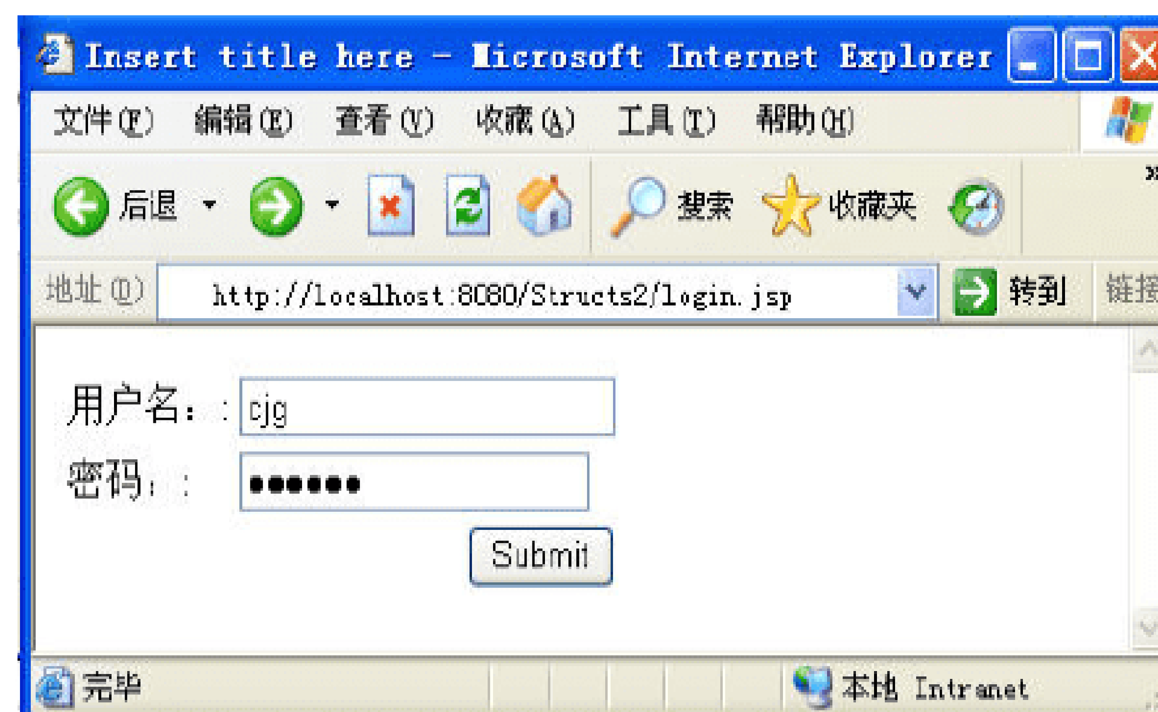


图 2.34　登录页面

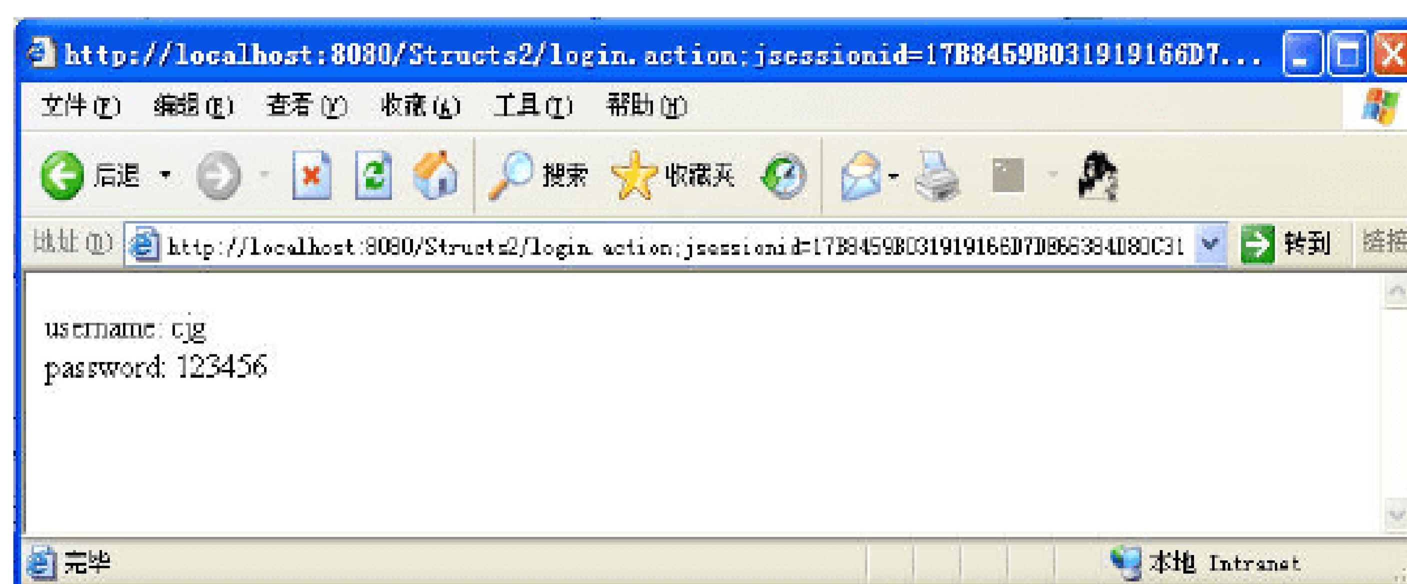


图 2.35　登录成功页面

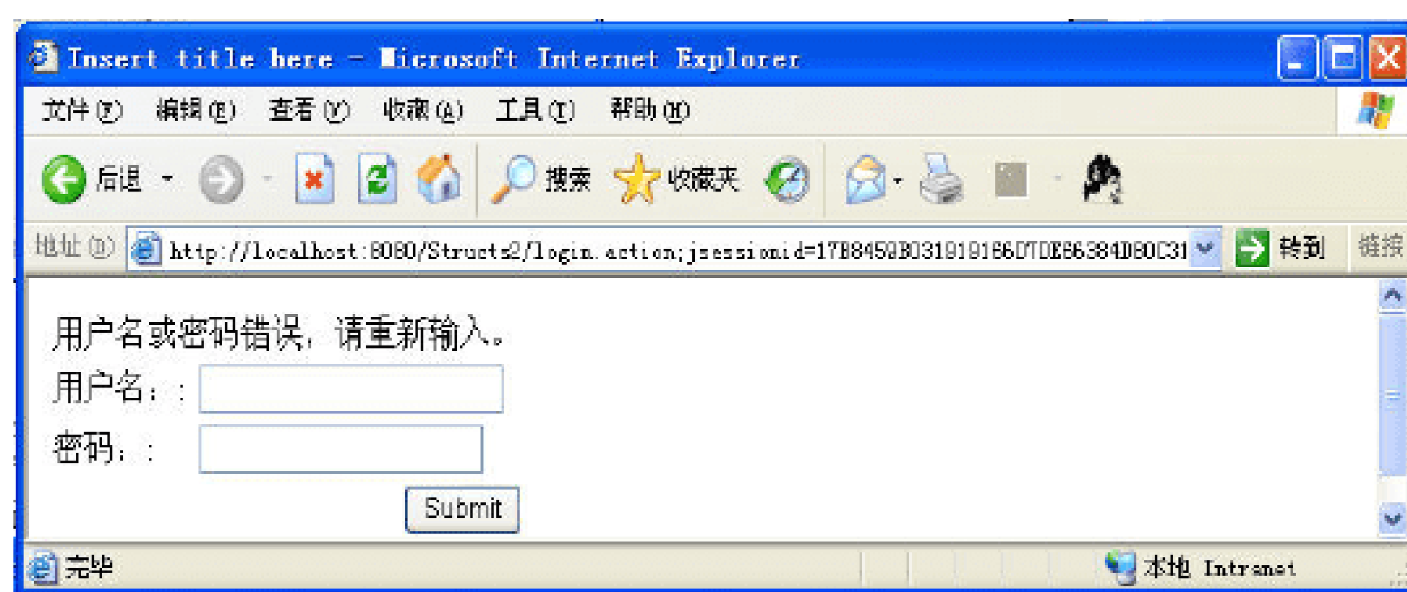


图 2.36　登录失败页面

2.2.7　分析 Struts 2.x 框架

通过 2.2.6 节介绍的开发步骤基本了解了 Struts 2.x 框架的运行过程，核心控制器 FilterDispatcher 会过滤所有请求，如果请求以 action 结尾，该请求将会转入框架处理。当框架获取*action 请求后，将根据*action 请求的前面部分决定调用哪个业务逻辑组件。最后根据业务逻辑组件的处理信息决定转发到哪个视图。

Struts 2.x 框架由 3 个部分组成：核心控制器 FilterDispatcher、业务控制器和业务逻辑

组件，其中核心控制器 FilterDispatcher 由 Struts 2.x 框架提供，而业务控制器和业务逻辑组件需要用户自己实现。

- 核心控制器 FilterDispatcher：负责拦截所有用户的请求，如果用户的请求以 action 结尾，该请求将被转入 Struts 2.x 框架处理。
- 业务控制器组件：实现 Action 类的实例，该类通常包含一个能返回一个字符串（逻辑视图名）的 execute()方法，用来实现项目的业务控制。
- 业务逻辑组件：跟 Struts 1.x 框架的业务逻辑组件一样，同样由 JavaBean 或 EJB 来实现。

Struts 2.x 框架中用于处理用户请求的并不是业务逻辑组件，而是 Action 代理。该过程是这样的：在 Struts 2.x 框架中存在一系列拦截器，这些拦截器将 HttpServletRequest 请求中的请求参数解析出来，传入到 Action 中，并回调 Action 的 execute()方法来处理用户请求。该过程如图 2.37 所示。

在知道了 Struts 2.x 框架的组成部分后，下面来讲解 Struts 2.x 框架的工作流程，该工作流程如图 2.38 所示。

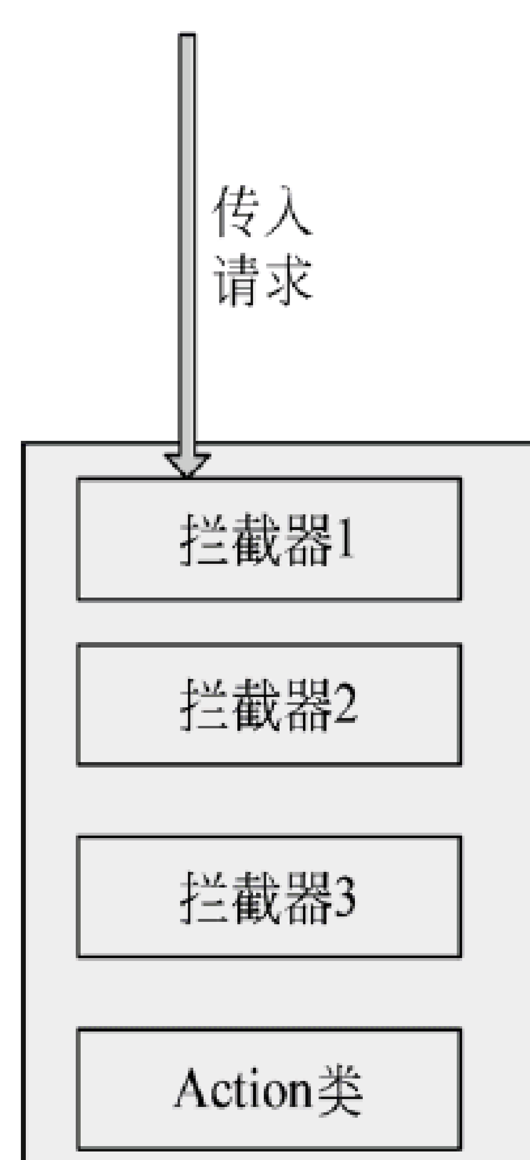


图 2.37　Action 代理

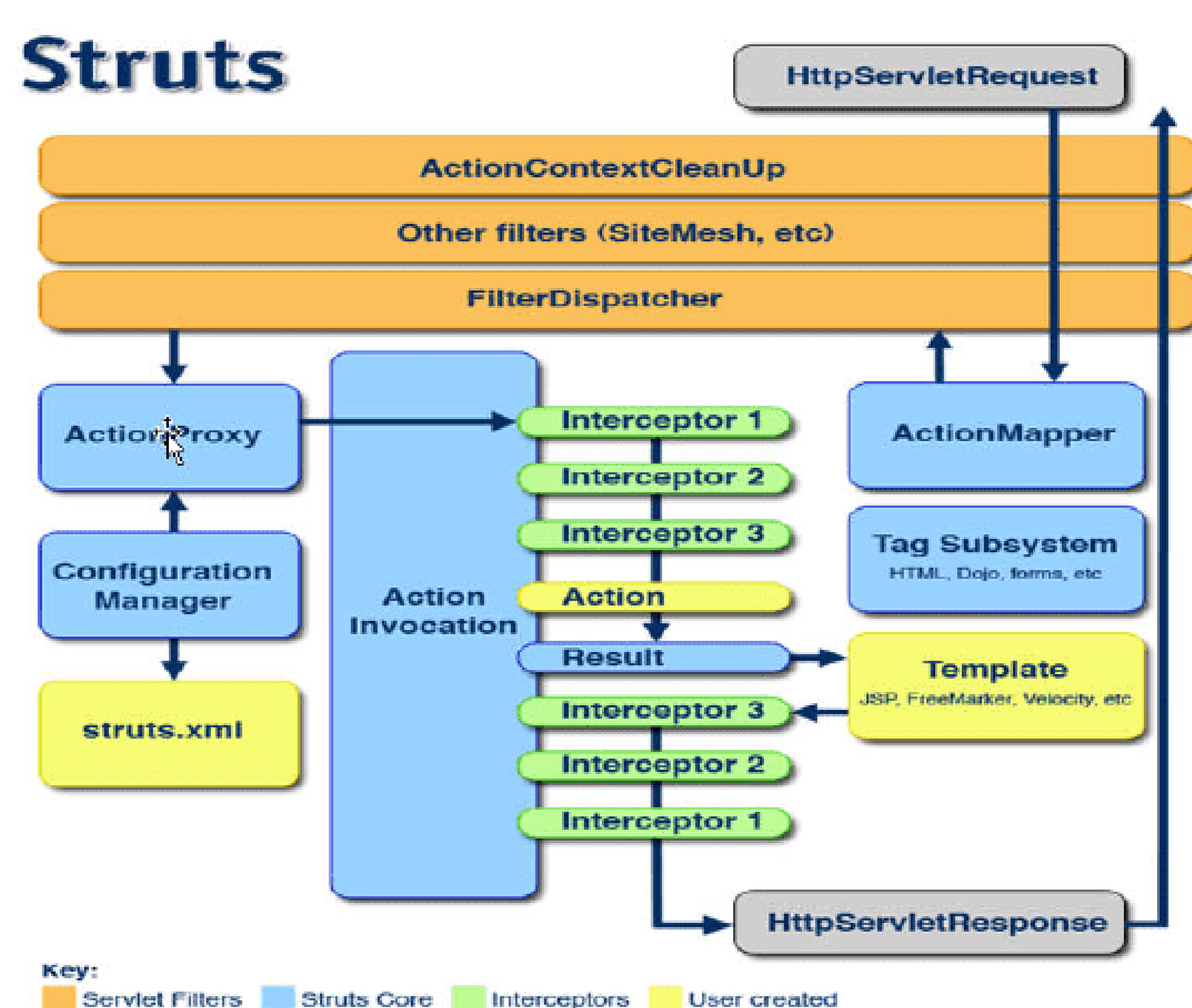


图 2.38　Struts 2.x 框架的工作流程

当所有请求被核心控制器 FilterDispatcher 拦截时，将执行如下流程。

（1）FilterDispatcher 会将请求转发给 ActionProxy（Action 代理），Action 代理会根据配置文件 struts.xml 决定转发给哪个 Action。

（2）在请求转发给 Action 过程中，会经过一系列拦截器，这些拦截器负责将请求解析并且转发给相应的 Action。

（3）经过相应 Action 的 execute()方法的处理，会得到一个视图名的结果。根据结果结合相应的模板产生相应的输出流。

（4）输出流也可以经过一系列的拦截器后，传送给浏览器。

2.3　Hibernate 框架的实现

为了让读者熟练地掌握基于 Hibernate 框架的应用，本节没有对 Hibernate 的持久化编

程和事务等概念做详细的讲解，而是具体讲解了如何使用开发环境 MyEclipse 来实现基于 Hibernate 框架应用的开发。

2.3.1　下载和了解 Hibernate 框架

到目前 Hibernate 最稳定的版本为 hibernate 3.2，可以通过访问 Hibernate 框架的官网 http://www.hibernate.org（如图 2.39 所示）来下载该版本的架包。

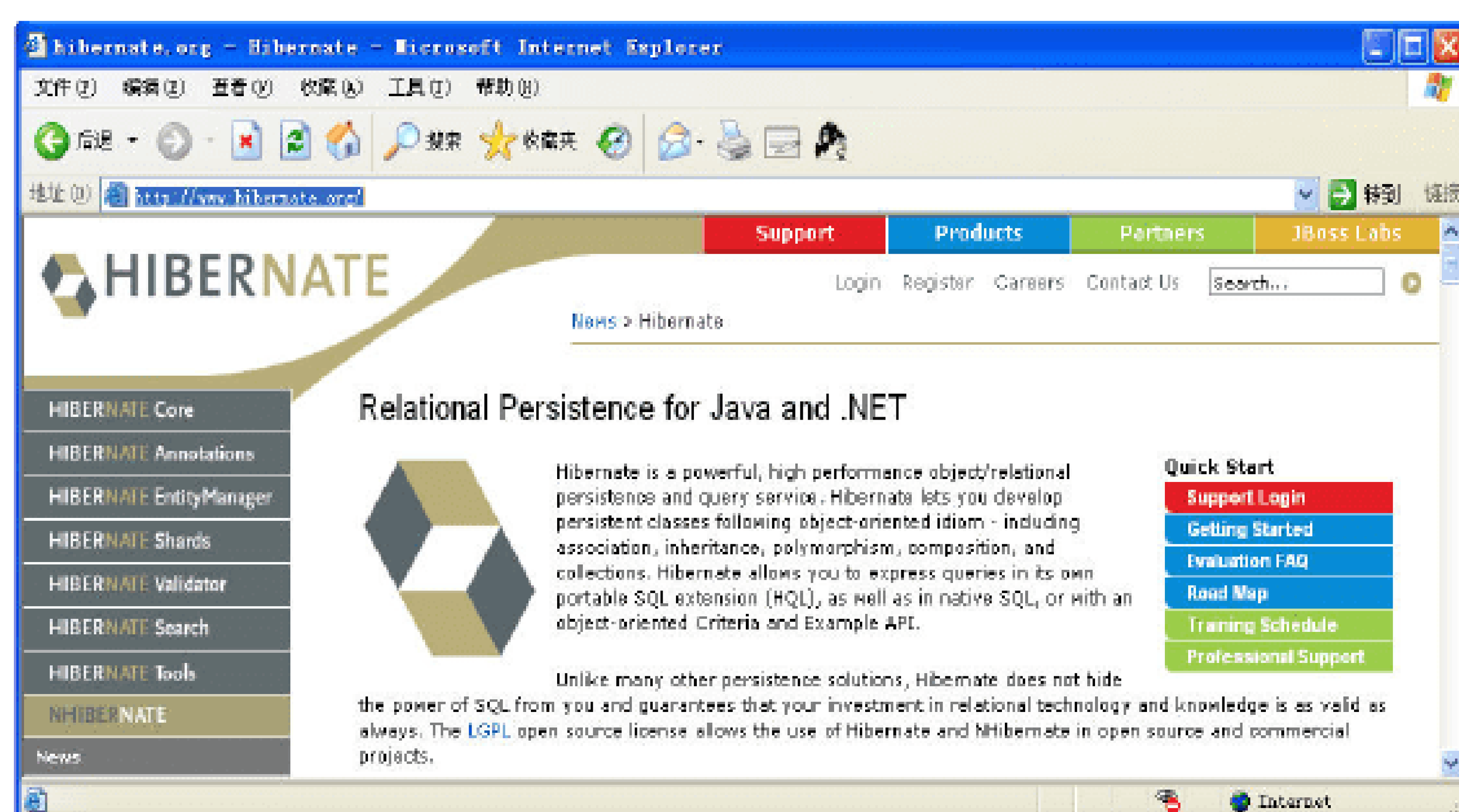


图 2.39　Hibernate 框架官网

下载完 hibernate-3.2 版后，将下载到的 Zip 文件解压，文件结构如下。

- hibernate3.jar：hibernate-3.2 的核心架包。
- doc：关于 hibernate-3.2 的相关文档。
- lib：关于 hibernate-3.2 的一些架包。
- src：关于 hibernate-3.2 的全部源代码。
- eg：关于 hibernate-3.2 的一个小事例。
- etc：关于 hibernate-3.2 的一些配置文件，在具体开发时可以参照它。
- test：关于 hibernate-3.2 的单元测试代码。
- build：分成 3 种类型：bat、sh 和 xml。xml 类型为 Ant 开发环境的脚本，bat 为 Windows 环境下的自动运行脚本而 sh 为 Unix 环境下的自动运行脚本。当它们在相应环境中运行时，会把 hibernate-3.2 架包整个编译。

如果想在 Java Web 项目中使用 hibernate-3.2，只要把 Hibernate 框架的核心架包复制到 Java Web 应用的 WEB-INF/lib 路径下就可以。

2.3.2　用 MyEclipse 实现 Hibernate 框架环境

为了便于讲解，本节将在 2.3.1 节完成的开发环境中，利用 Hibernate 框架实现一个具体的应用。首先介绍代码的运行背景，用户通过用户名和密码实现登录功能，具体步骤如下。

（1）新建一个名 Hibernate 的 Java Project，详细设置如图 2.40 所示。

（2）为项目增加 Hibernate 框架的相关类库与文件。在 Package Explorer 视图中右击该项目，在弹出的快捷菜单中选择 MyEclipse|Project Capabilities|Add Hibernate Capabilities 命令（如图 2.41 所示），打开 Add Hibernate Capabilities 对话框。

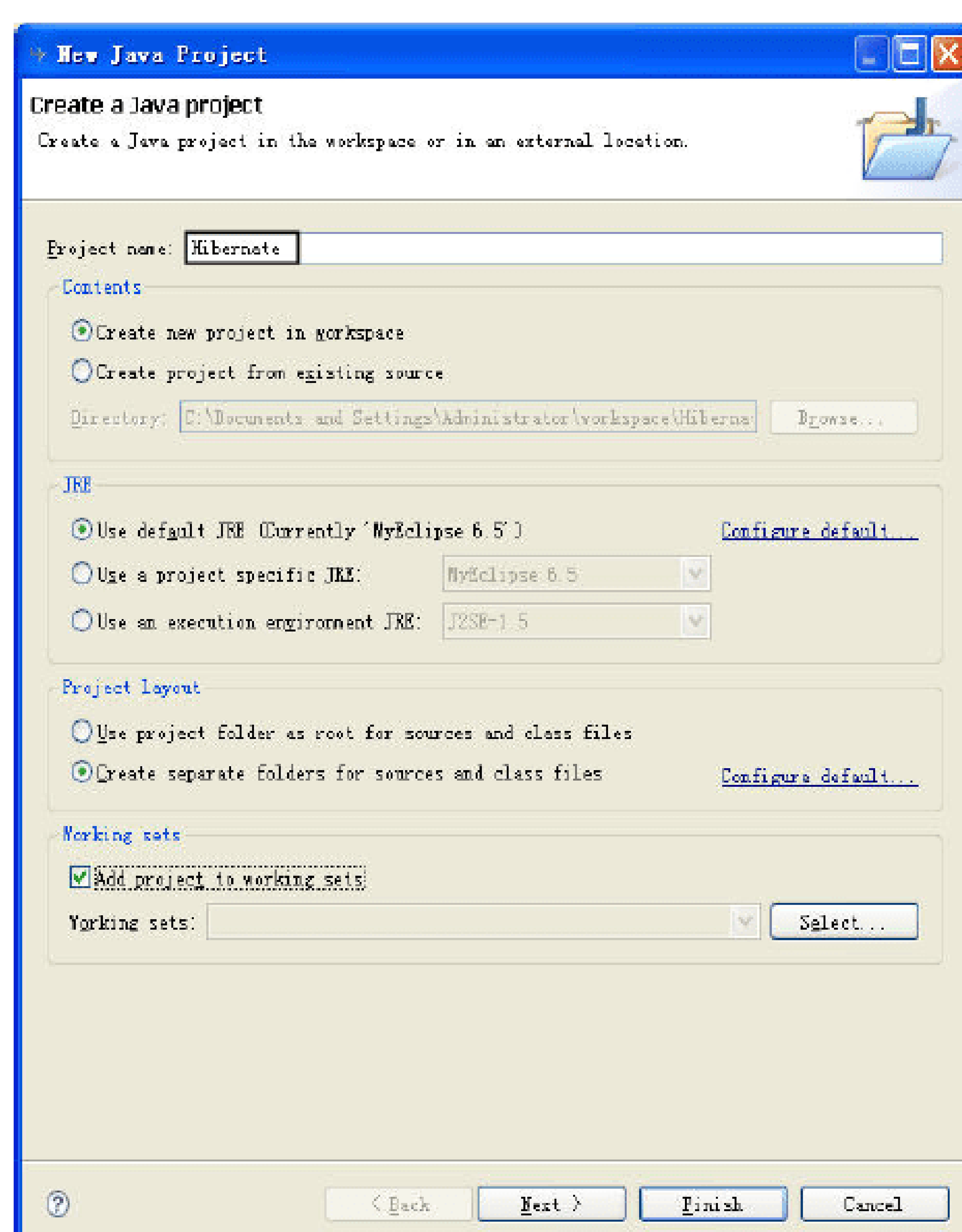


图 2.40　新建 Java Project

注意：该步骤也可以通过选择 MyEclipse|Project Capabilities|Add Hibernate Capabilities 命令来实现，如图 2.42 所示。

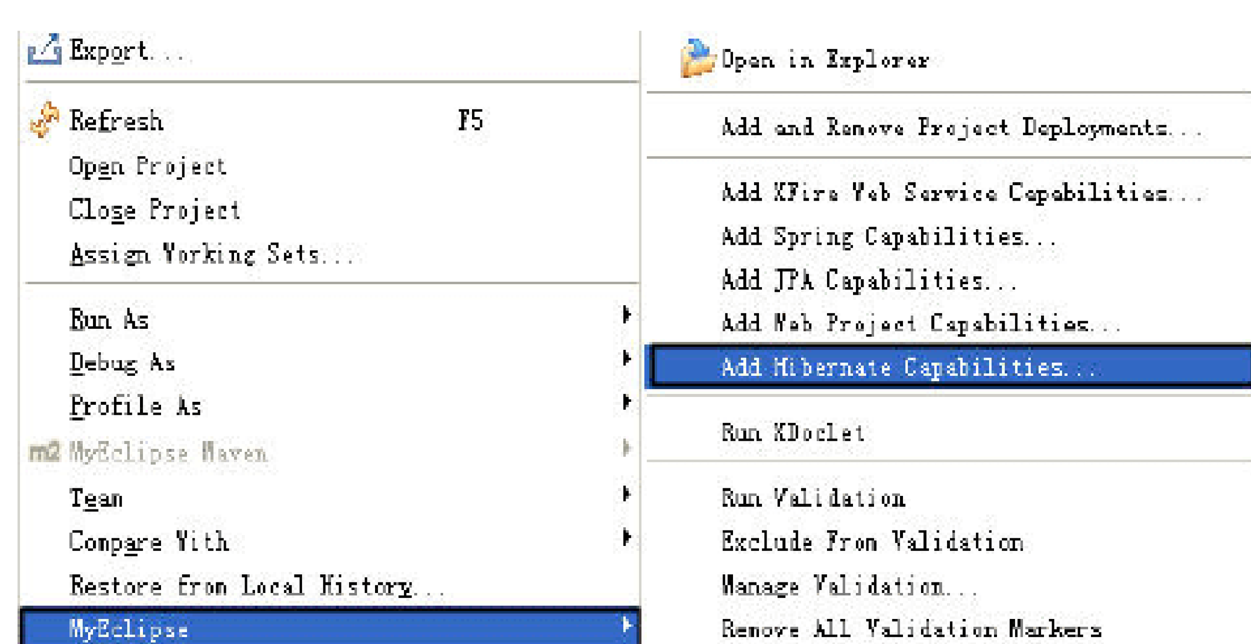


图 2.41　通过右键实现添加 Hibernate

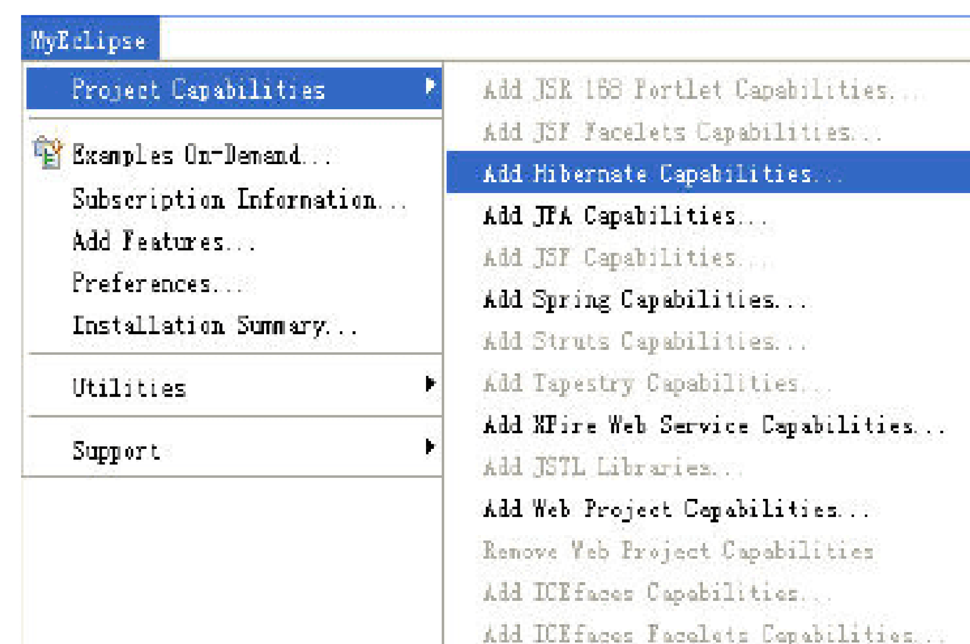


图 2.42　通过菜单实现添加 Hibernate

在如图 2.43 所示的 Hibernate Support for MyEclipse（添加 Hibernate 功能）对话框中，一般选默认值就可以。当然也可以根据需要通过修改一些地方来定制将来生成的类。各个选项的具体含义如下。

- Hibernate Specification：用来选择 Hibernate 的版本，在该项目选择 Hibernate 3.0。
- Select the libraries to add to the buildpath：用来设置要加入项目类路径的类库。

注意：Hibernate 框架的类库是按照模块进行划分的，所以可以根据项目的需要选择必要的类库，而不需要选择全部的类库。

- JAR Library Installation：指定 JAR 类库的安装方式。上面的单选按钮表示把引用的类库加入类路径，但是相应的 JAR 文件将不会复制到当前项目中；下面的单选按钮则是把指定目录下的所有的 JAR 文件和元素文件复制到当前项目下。
- Library Folder：相对于现在项目的路径，可以新建或者使用一个新的目录，Hibernate 类库将会被向导复制到该新的目录下。

接着单击 Next 按钮，弹出如图 2.44 所示的创建 Hibernate 配置文件对话框。

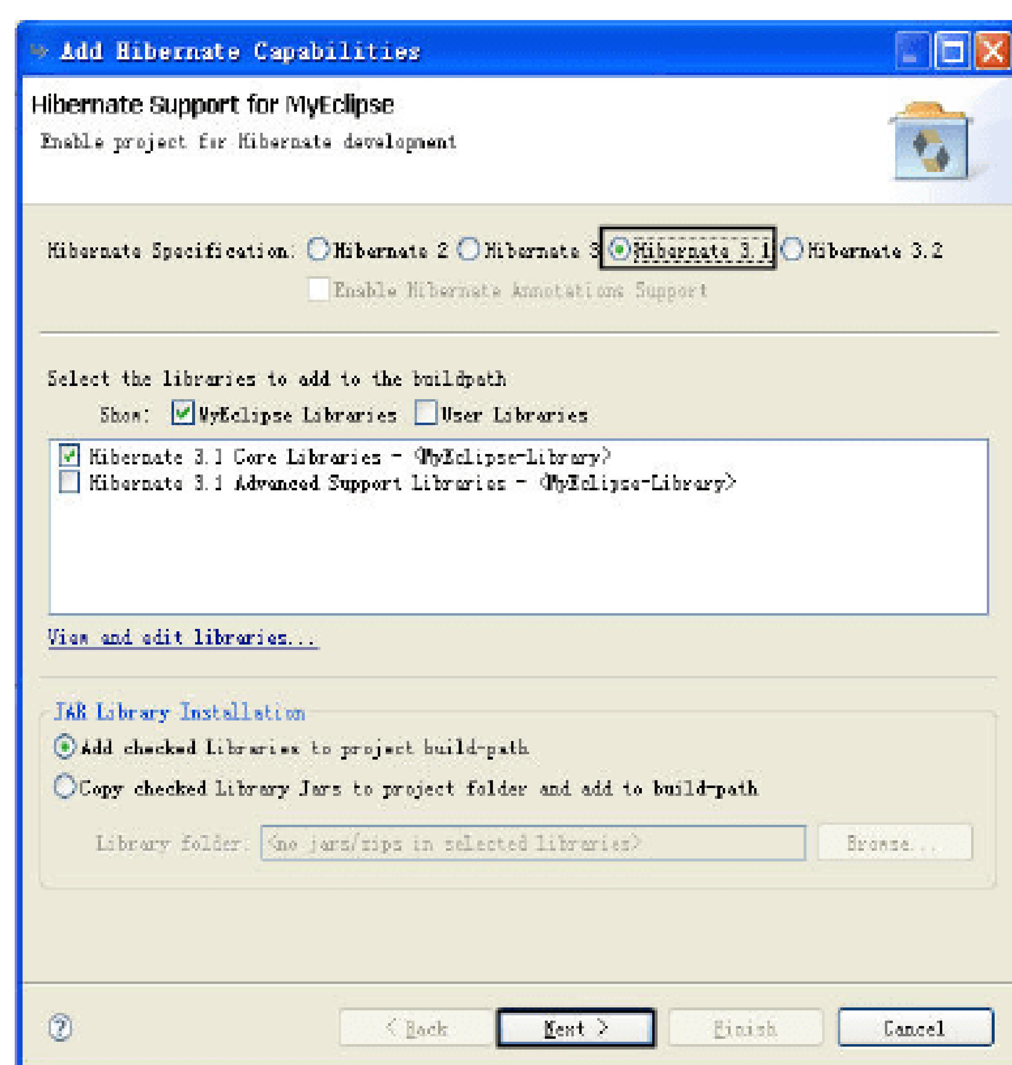


图 2.43　添加 Hibernate 框架

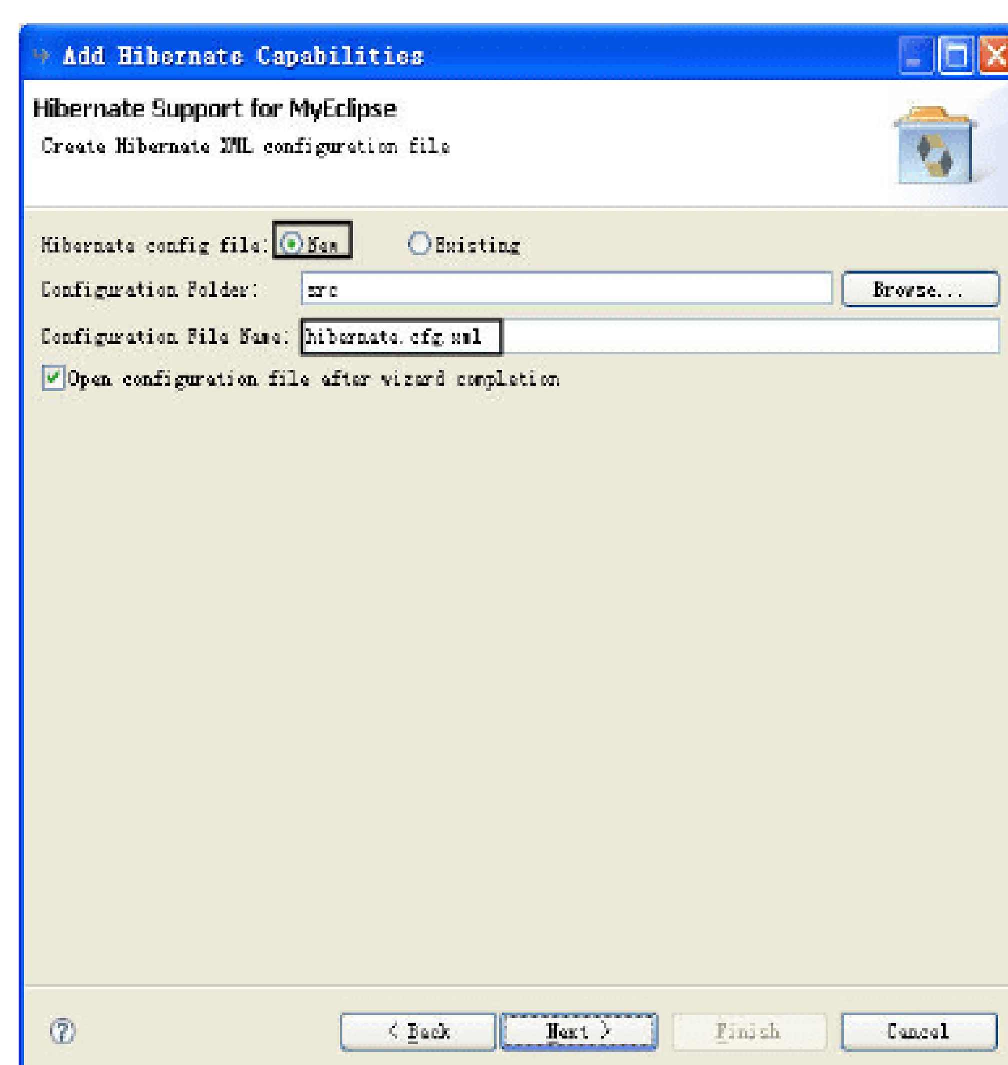


图 2.44　创建 Hibernate 配置文件

如果是全新项目保持默认设置就可以，如果想使用以前存在的 Hibernate 配置文件，可以选择 Existing 单选按钮，然后选择 Hibernate 配置文件的路径就可以。接着再次单击 Next 按钮，弹出选择 Hibernate 所使用数据库连接的对话框，如图 2.45 所示。单击 DB Driver 右侧的现有数据库连接列表，选择创建好的数据库 MySQL，其他相关的连接信息将会自动填入到对话框中。

注意：当选择 Copy DB driver jar(s) to project and to buildpath? 复选框后，则会自动添加数据库驱动类库 jar 文件到项目的类路径中。

（3）创建 SessionFactory 类，通过再次单击 Next 按钮，就会弹出 Create Hibernate SessionFactory（创建 SessionFactory）对话框，如图 2.46 所示。

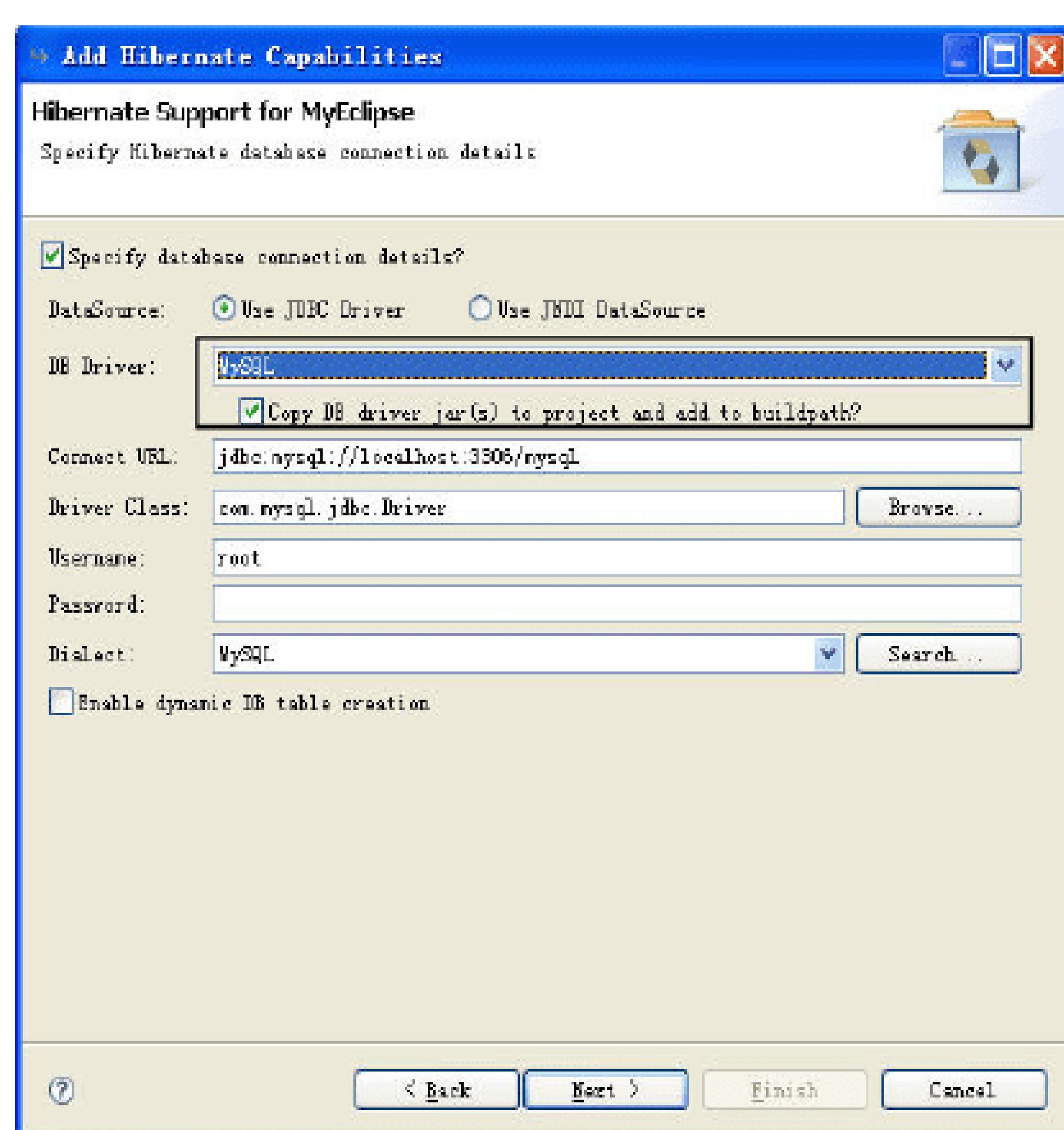


图 2.45　选择 Hibernate 所使用的数据库连接

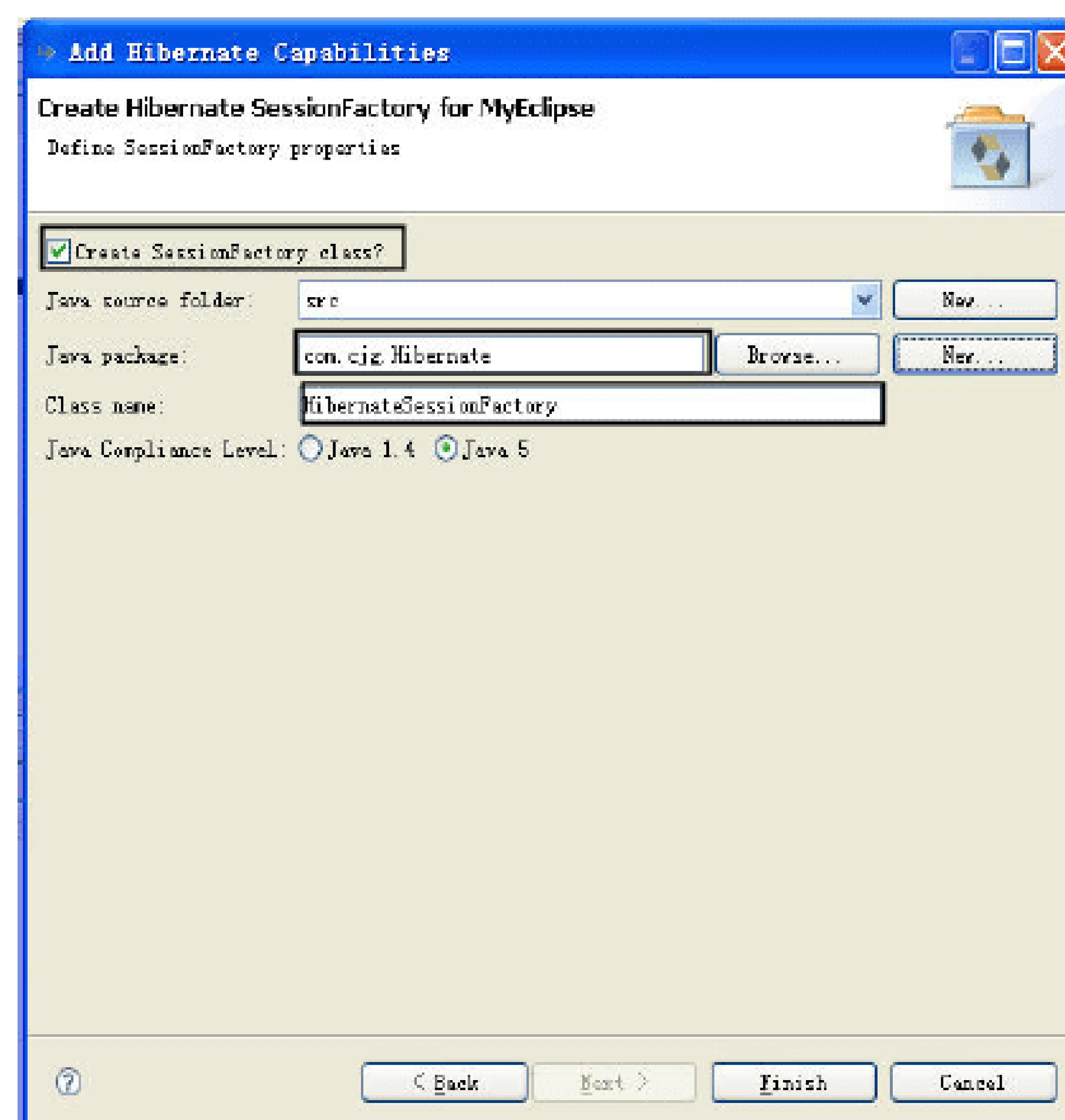


图 2.46　创建 SessionFactory

如果没有存在的包，则单击 Java package 输入框右侧的 New 按钮创建一个包，如图 2.47 所示。

注意：这一页的设置是可选的，如果现在不想创建 SessionFactory 类，取消 Create SessionFactory class 复选框的选择就可以跳过。

当完成增加 Hibernate 相关类库与文件操作后，展开 Package Explorer 视图中 Hibernate 项目目录后，如图 2.48 所示。

- Hibernate 3.0 Core Libraries：为项目添加 Hibernate 类库。
- hibernate.cfg.xml：为项目添加的 Hibernate 配置文件。
- HibernateSessionFactory.java：获取 Hibernate 会话的工具类，会自动加载 Hibernate 的配置文件。

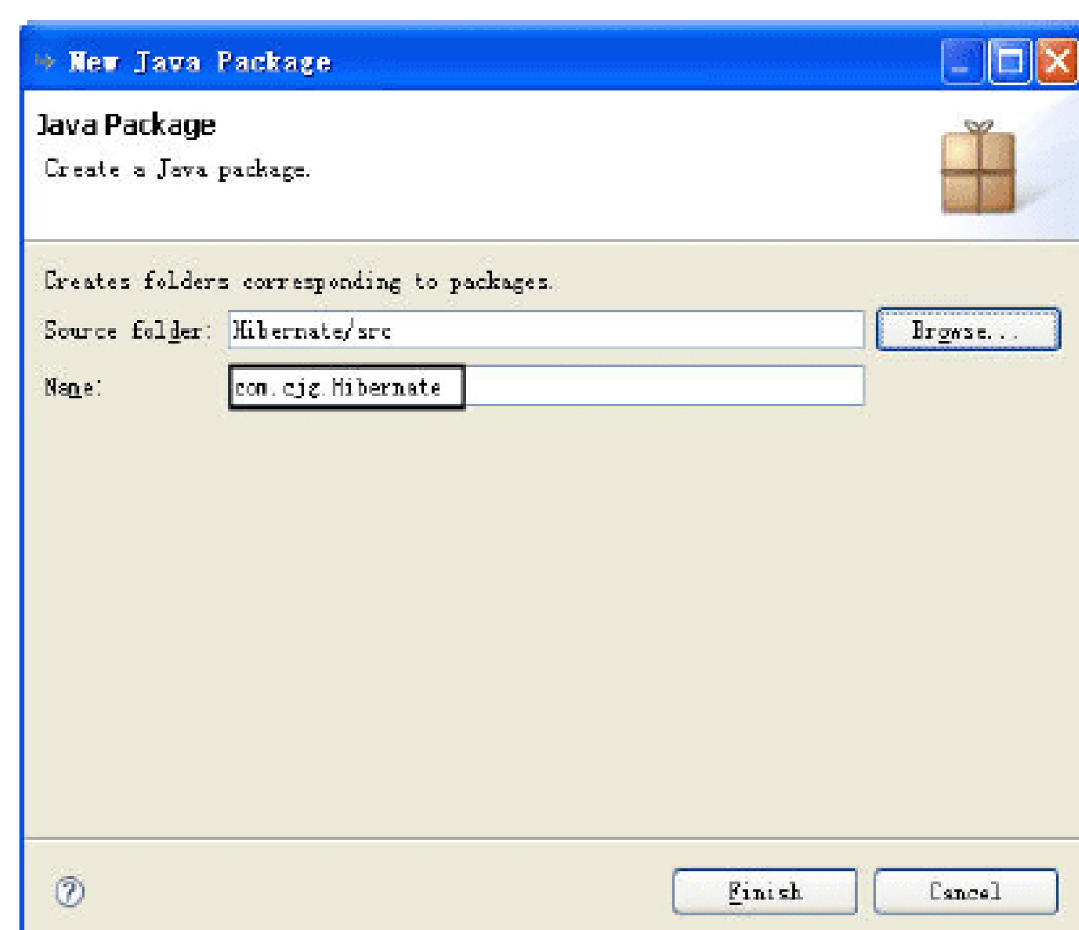


图 2.47　创建 Java package

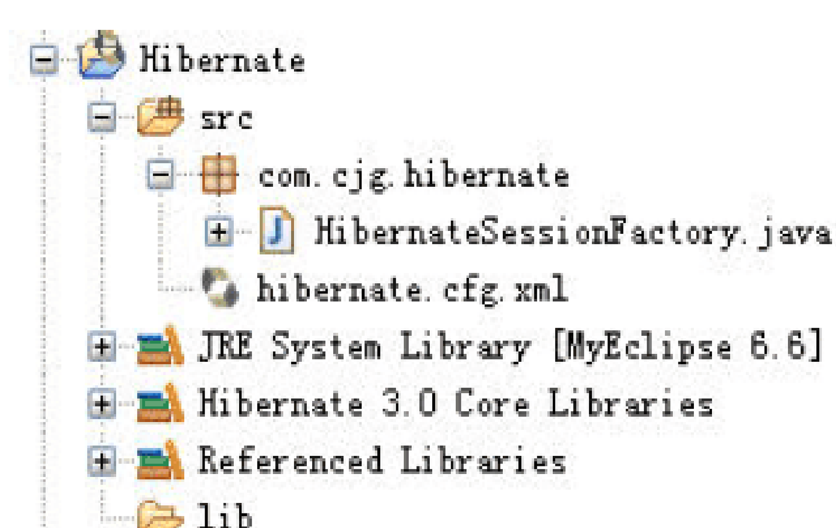


图 2.48　Hibernate 项目目录

至此，该项目已经具备了 Hibernate 框架的支持。

2.3.3　MyEclipse 对 Hibernate 框架支持——关系数据库到对象映射

为了便于讲解，本节将在 2.3.2 节完成的开发环境中，利用 Hibernate 框架实现一个具体的应用。首先介绍代码的运行背景，读取数据库表格中的信息，即把数据库表格中的信息转化为对象关系然后打印出来，具体步骤如下。

在介绍之前，先熟悉一下 Hibernate 配置文件编辑器，通过双击 hibernate.cfg.xml 文件，可以打开 Hibernate 配置文件编辑器，如图 2.49 所示。



图 2.49　Hibernate 配置文件编辑器

（1）首先根据数据库表生成 POJO 类和映射文件，实际上就是把数据库中的表格信息转变成普通的 Java 对象，MyEclipse 对该功能提供了支持。该功能的实现是在 DB Browser

视图中实现。

注意：POJO 全称 Plain and Old Java Object，普通和旧式的 Java 对象的缩写，也就是普通 Java 类的意思。

首先介绍一下 DB Browser 视图，通过单击按钮弹出如图 2.50 所示的快捷菜单。选择 MyEclipse Hibernate 选项，出现 DB Browser 视图，如图 2.51 所示。

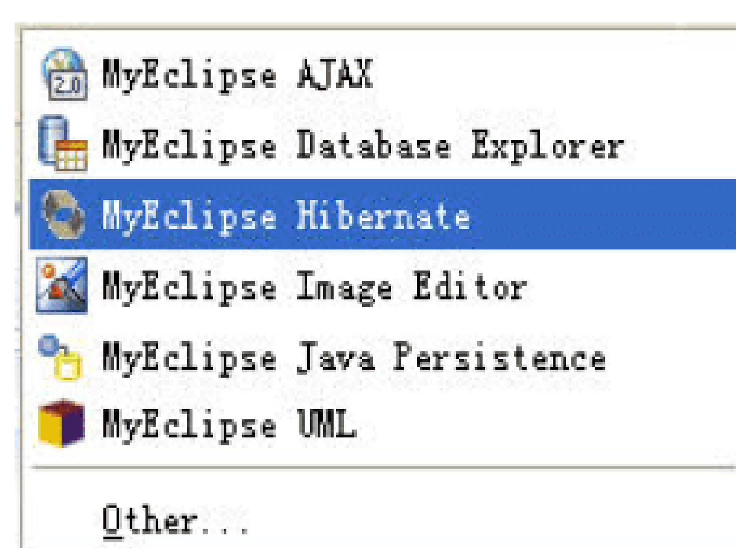


图 2.50　快捷菜单

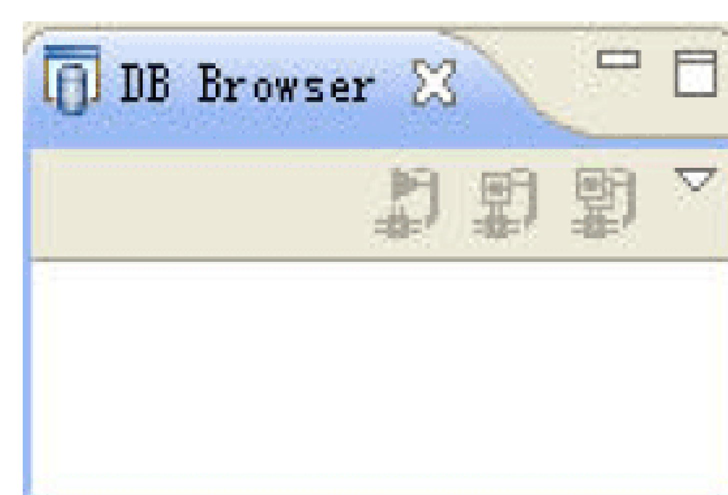


图 2.51　DB Browser 视图

右击 DB Browser 视图，在弹出的快捷菜单中选择 New 命令（如图 2.52 所示），打开如图 2.53 所示的 Edit Database Connection Driver（配置数据库链接）对话框。在该对话框中，一般首先通过 Driver template 选项来设置驱动模块，因为其他选项可以根据该选项给出相应的提示。

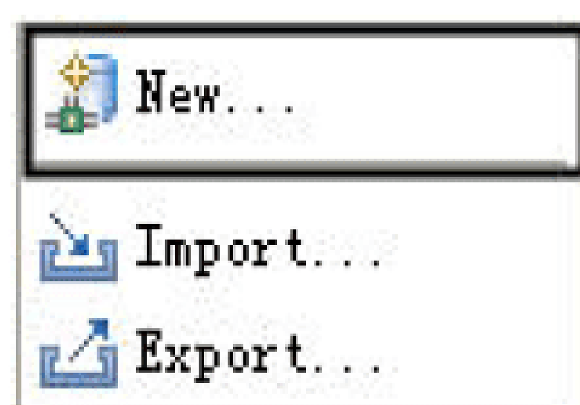


图 2.52　快捷菜单

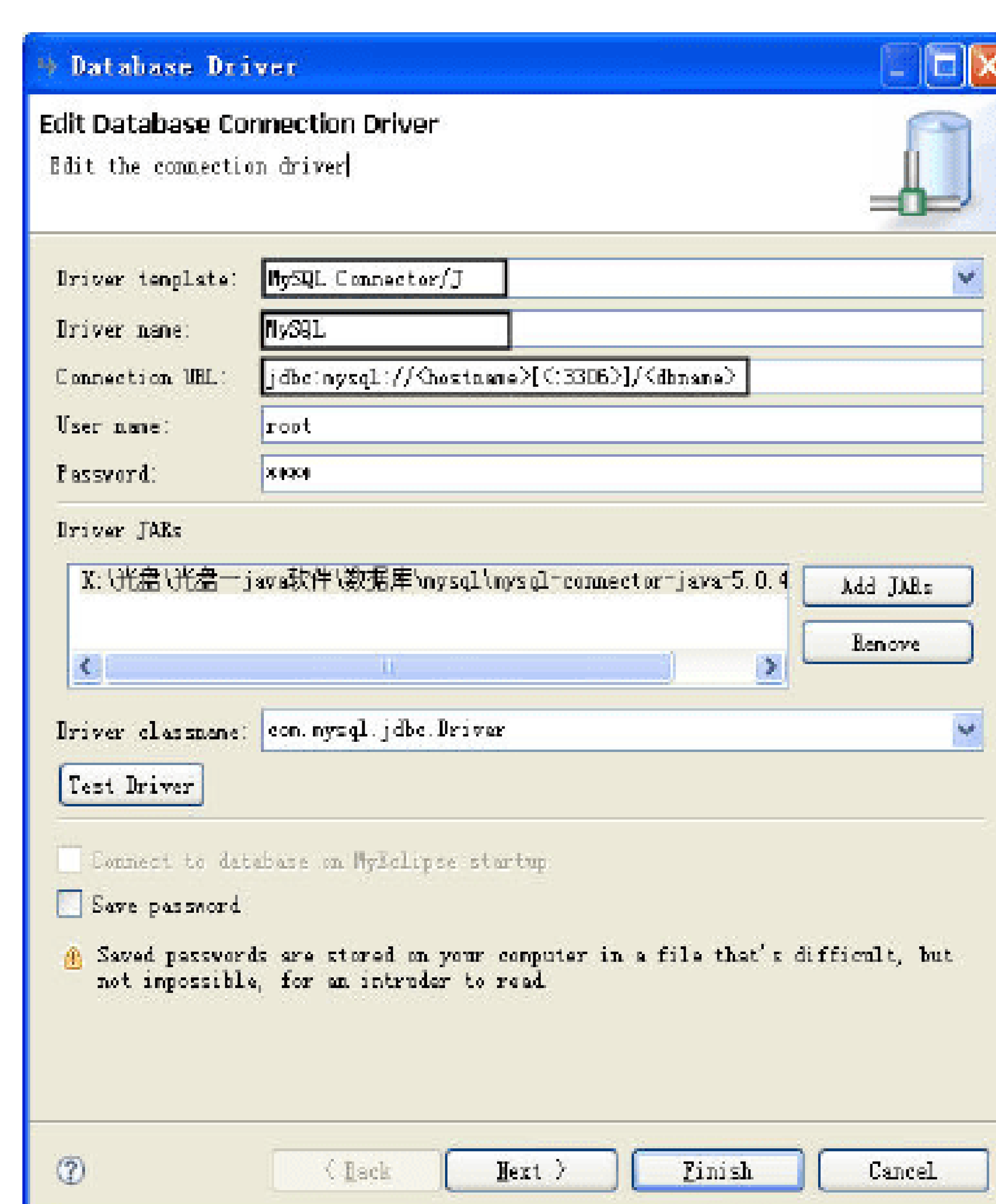


图 2.53　配置数据库连接

各个选项的具体含义如下。

- Driver template：用来设置驱动模板，在这里设置为 MySQL Connector/J。
- Driver name：用来设置驱动名字，可以任意设置一个名字，这里设置为 MySQL。
- Connection URL：用来设置链接地址，在这里设置为 jdbc:mysql://localhost/mysql。
- User name 和 Password：分别用来填写用户名和密码。
- Driver JARs：通过单击 Add JARa 按钮来添加驱动包。

注意：只有通过该方式配置好的数据库连接，才会在图 2.45 中 DB Driver 右侧的现有数据库连接列表中出现。

配置完名为 MySQL 的数据库连接后，在 DB Browser 视图中展开树状表结构，选择要持久化的 student 数据库中 student 表格。右击该表格，在弹出的快捷菜单中选择 Hibernate Reverse Engineering 命令（如图 2.54 所示），打开 Hibernate 逆向工程向导（如图 2.55 所示）。

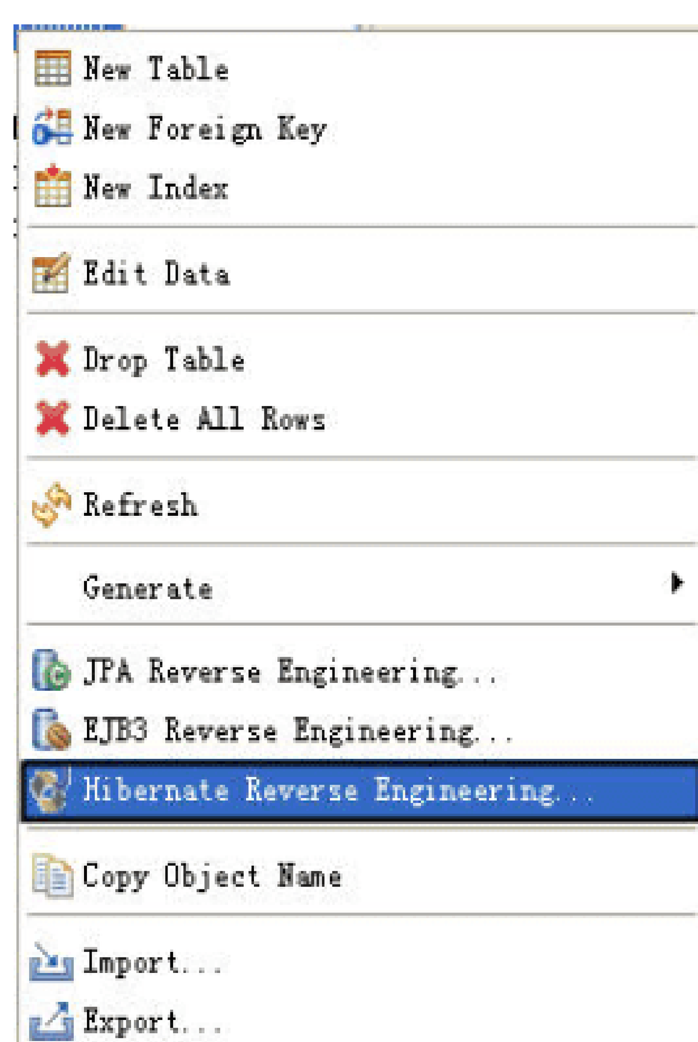


图 2.54　逆向工程菜单

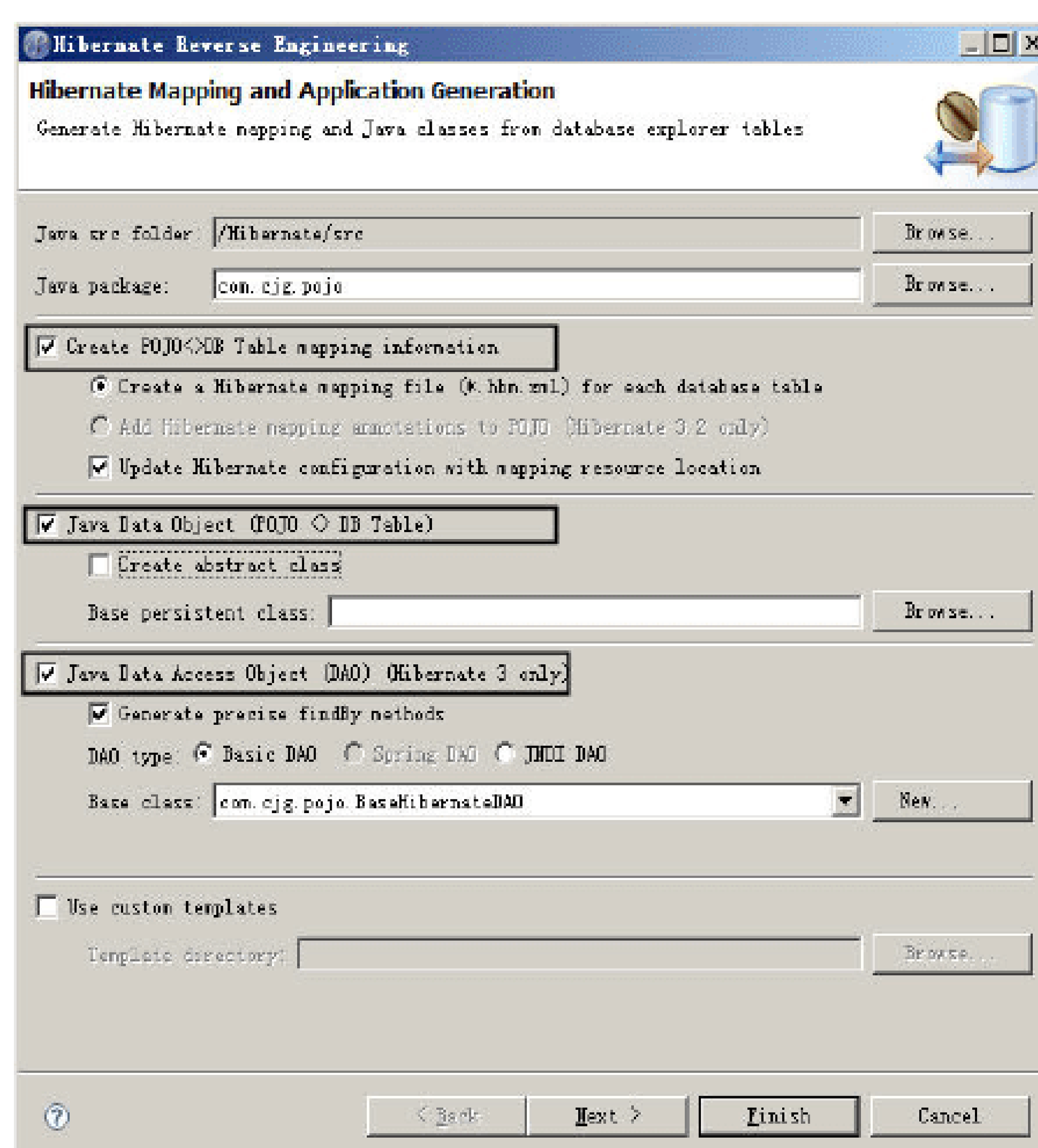


图 2.55　逆向工程对话框

各个选项的具体含义如下。

- Java src folder：POJO 和 DAO 生成后源代码的文件夹。
- Java package：POJO 和 DAO 生成后所在包。
- Create POJO DB Table mapping information：为选中的表格生成映射文件。
- Java Data Object：为每个映射文件和表格生成对应的数据对象（POJO）。

注意： 虽然 MyEclipse 对逆向功能提供帮助，但是一定要对生成源代码进行进一步的检查、调整和修改。

当完成逆向工程的操作后，展开 Package Explorer 视图中 Hibernate 项目目录，如图 2.56 所示。

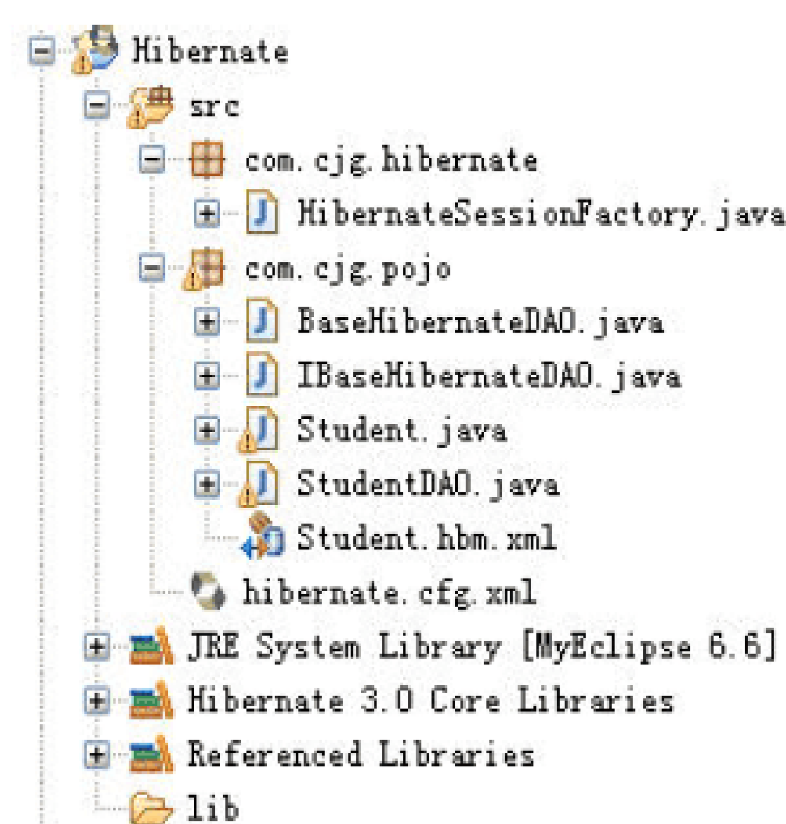


图 2.56　Hibernate 项目目录

在自动生成的文件中，只需要注意 3 个文件即可。

- Student.java：DAO 层实体类。
- StudentDAO.java：DAO 层核心类。
- Student.hbm.xml：实体类映射文件。

对于 HibernateSessionFactory.java 文件，只需要调用 getSession()方法就可以获得一个 Session 对象，调用 closeSession()方法则关闭当前的 Session 对象。对于 IBaseHibernateDAO.java 则定义了一个接口，用来获取 Session 对象的操作，而 BaseHibernateDAO.java 则实现了该接口，通过使用 HibernateSessionFactory 获取会话。最后，StudentDAO 类则继承了上面的接口，并实现了对 Student 实体类的增、删、改、查（CRUD）方法。在具体编写时，可以直接调用 StudentDAO 类。

（2）编辑测试类。在 Hibernate\src 目录中新建一个名为 test 的 Java 类，用来读出数据库表格中的信息。代码 2.10 实现了数据库表中信息的输出。

代码 2.10　测试类：test.java

```
...
public class test {
    public static void main(String[] args) {
        StudentDAO dao = new StudentDAO();                    //实例化 DAO
        java.util.List<Student| results = dao.findAll();   //读取数据
        for (Student o : results) {                       //列出列表中的所有数据
            System.out.println("编号:" + o.getId());
            System.out.println("姓名:" + o.getName());
            System.out.println("学号:" + o.getNumber());
        }
        dao.getSession().close();                         //关闭 DAO
    }
}
```

（3）在运行项目前，先查看一下 student 数据库中 student 表格的信息，如图 2.57 所示。当编译 test.java 后，其运行结果如图 2.58 所示。

	id	name	number
	1	cjg	1
*	(NULL)	(NULL)	(NULL)

图 2.57　数据表格信息

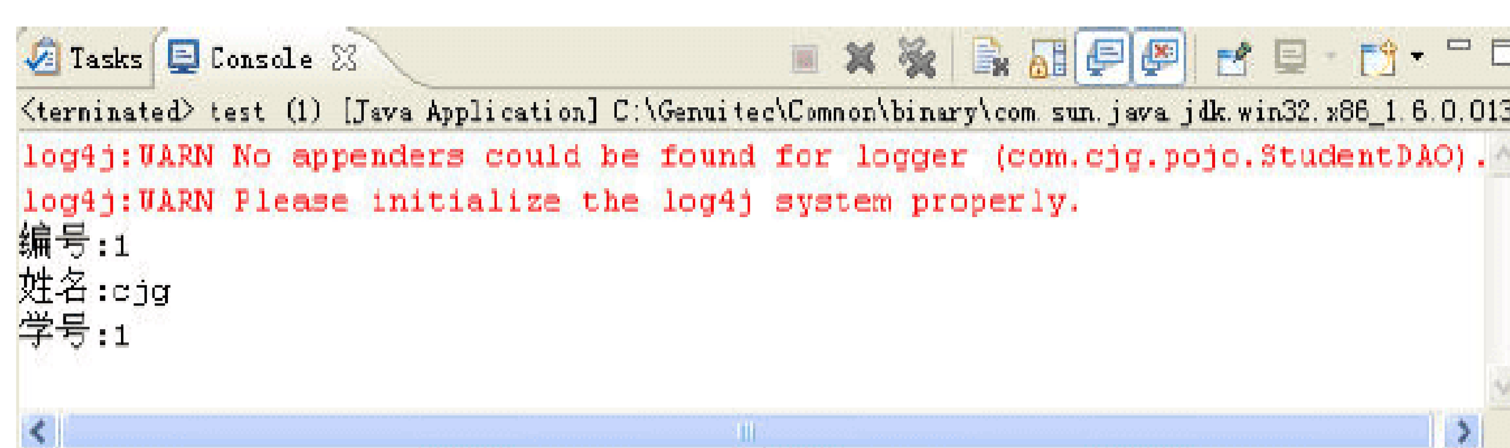


图 2.58　运行结果

2.3.4　Hibernate 框架中经常用到的工具类

2.3.2 节关于 Hibernate 框架应用中，是把关系模型（数据库中表格信息）转换成对象模型（Java 对象），而在本节将实现把对象模型（Java 对象）转换成关系模型（数据库中表格信息）的工具类，并实现把对象中的信息写进数据库中表格的工具类。具体步骤如下。

（1）首先通过向导新建一个名为 HibernateTool 的 Java Web 项目，然后通过向导使当前的项目支持 Hibernate 框架应用。在添加 Hibernate 框架向导的第一页和第二页，保持默认选项就可以。接着再单击 Next 按钮进入向导的第三页，在这一页只要在 DB Driver 选项中选择创建好的数据库连接 MySQL 就可以配置好数据库连接，如图 2.59 所示。接着再次单击 Next 按钮进入向导的第四页，在该页中取消 Create SessionFactory class 复选框的选择，最后单击 Finish 按钮完成添加 Hibernate 框架到项目的过程。完成支持 Hibernate 框架开发的环境后，其目录结构如图 2.60 所示。

（2）在 HibernateTool/src 文件夹下，新建一个名为 com.cjg.hibernate 的包。然后在该包中建立一个名为 User 的 JavaBean 对象，最后为该对象建立映射文件。代码 2.11 创建了一

个实体类，代码 2.12 为实体类 User 的映射文件。

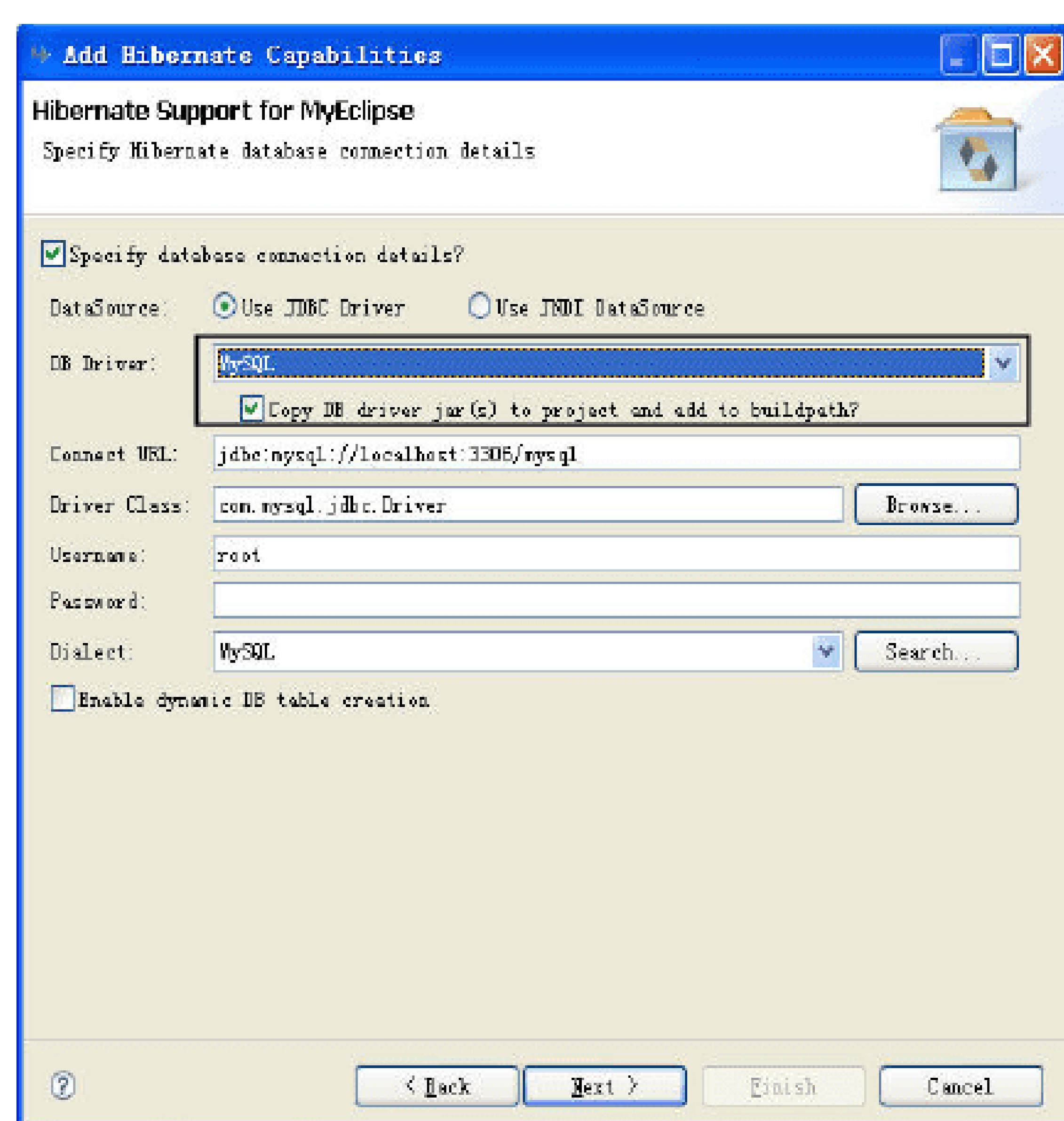


图 2.59　配置数据库连接

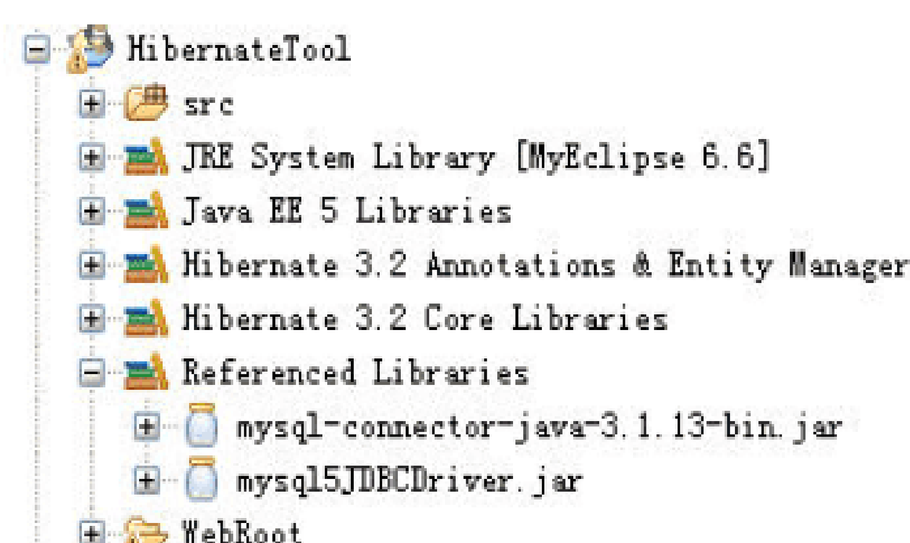


图 2.60　目录结构

代码 2.11　user 实体类：user.java

```
...
public class user {
    private String id;                          //定义了一个 id 属性
    private String name;                        //定义了一个 name 属性
    private String password;                    //定义了一个 password 属性
    //省略相关属性的 get()和 set()方法
...
}
```

代码 2.12　映射文件：user.hbm.xml

```
...
<hibernate-mapping>
    <class name="com.cjg.hibernate.user">  <!--用来设置实体类-->
        <id name="id">                      <!--用来设置表格的主建字段-->
            <generator class="uuid"/>
        </id>
        <property name="name"/>             <!--用来设置表格的普通字段-->
        <property name="password"/>
    </class>
</hibernate-mapping>
```

【代码解析】

- 元素<class>的 name 属性值是实体类的完整路径，默认情况下该实体类映射表格为实体类的名字，也可以通过属性 table 设置。
- 子元素<id>用来把实体类中的标识转换成数据库表格中的主键，该子元素中的 name 属性值是实体类中的标识，默认转换成的表格中的字段名为标识名，也可以通过属性 column 来设置。对于主键的生成策略，则通过元素<generator>来设置。
- 子元素<property>用来把实体类中的普通属性转换成数据库表格中的字段。

注意： 实体类与该实体类的映射文件一般放在同一个包。

（3）创建一个用来把对象模型转变成关系模型的工具类 ExportDB.java，代码 2.13 用来实现把实体类转换为一个数据库表。

代码 2.13　转换工具类：ExportDB.java

```
...
public class ExportDB {
    public static void main(String[] args) {
        Configuration cfg = new Configuration().configure();
                                   //读取 hibernate.cfg.xml 文件
        SchemaExport export = new SchemaExport(cfg);
                                   //获取实现转换的类 SchemaExport
        export.create(true, true);         //执行转换
    }
}
```

（4）生成数据库表，先创建一个名为 Hiber 的数据库，然后运行 ExportDB.java 文件。查看 Hiber 数据库则出现包含如图 2.61 所示字段的表。

	▲ id	name	password	createTime	expireTime
*		(NULL)	(NULL)	(NULL)	(NULL)

图 2.61　生成的表格

为什么是 Hiber 数据库呢？为什么会在 Hiber 数据库中创建出字段呢？这是因为在 hibernate.cfg.xml 配置文件中做了配置，代码 2.14 为 Hibernate 配置文件。

代码 2.14　Hibernate 配置文件：hibernate.cfg.xml

```
<hibernate-configuration>
   <session-factory>
      <property name="connection.username">root</property>
      <!-- connection.url 最后的设置则为数据库名-->
      <property name="connection.url">jdbc:mysql://localhost/Hiber
      </property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect
      </property>
      <property name="myeclipse.connection.profile">MySQL</property>
      <property name="connection.password">root</property>
      <property name="connection.driver_class">com.mysql.jdbc.Driver
      </property>
      <!--设置实体类的映射文件-->
      <mapping resource="com/cjg/hibernate/User.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

【代码解析】

在上述代码中 name 值为 connection.url 的<property>元素用来设置数据库 URL，即 jdbc:mysql://localhost/Hiber，其中 Hiber 为所要建的数据库名。

（5）根据实体类创建出数据表后，如何把数据输入到数据表中呢？在实现该功能时，必须要用到 SessionFactory、Session 等对象，但是 SessionFactory 是一个重量级对象创建时需要很长时间，所以最好把 SessionFactory 和 Session 对象封装起来成为一个工具类。代码 2.15 实现了 SessionFactory 和 Session 两个对象的封装。

代码 2.15　封装重要对象：HibernateUtils.java

```
...
public class HibernateUtils {
    private static SessionFactory factory;
                                //定义了一个静态的 SessionFactory 对象
    //SessionFactory 对象只需要初始化一次，应该在静态块中初始化该对象
    static {
        try {
            Configuration cfg = new Configuration().configure();
                                                //读取配置文件
            factory = cfg.buildSessionFactory();   //初始化对象 factory
        }catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static SessionFactory getSessionFactory() { //获取对象 factory
        return factory;
    }

    public static Session getSession() {                //获取对象 Session
        return factory.openSession();                   //创建对象 Session
    }

    public static void closeSession(Session session) { //关闭对象 Session
        if (session != null) {
            if (session.isOpen()) {
                session.close();                        //关闭对象 Session
            }
        }
    }
}
```

（6）最后在包 com.cjg.hibernate 中新建一个名为 add 的测试类，该类利用 HibernateUtils 工具类来实现把信息输入到数据表中。代码 2.16 用来实现向数据库表格中添加信息。

代码 2.16　添加信息：add.java

```
...
public class add {
    public static void main(String[] args) {
        Session session = null;         //定义一个 hibernate 的 Session 对象
        User user1 = null;              //定义一个用户对象
        try {
            session = HibernateUtils.getSession();
                                        //通过工具类获取 Session 对象
            Transaction tx = session.beginTransaction();
                                        //Session 对象绑定到当前事务
            user1 = new User();         //新建一个用户对象
            user1.setName("cjg");       //设置用户名
            user1.setPassword("cjg");   //设置密码
            session.save(user1);        //保存对象到 Session 对象
            tx.commit();                //提交事务
        }catch(Exception e) {
            e.printStackTrace();
            tx.rollback();              //回滚事务
```

```
        }finally {
            HibernateUtils.closeSession(session);  //关系 Session 对象
        }
    }
}
```

（7）利用实体类向数据库表中填写信息，运行 add.java 文件后，查看 Hiber 数据库中的表 user，则包含了如图 2.62 所示的字段值。

	id	name	password	createTime	expireTime
□	4028808222d4ddc00122d4ddc2c90001	cjg	cjg	(NULL)	(NULL)
*		(NULL)	(NULL)	(NULL)	(NULL)

图 2.62　运行结果

2.4　JPA 框架的实现

为了让读者熟练地掌握基于 JPA 框架的应用的开发，本节没有对 JPA 框架的核心概念做详细的讲解，而是具体的讲解了如何使用开发环境 MyEclipse 来实现 JPA 框架应用的开发。

2.4.1　用 MyEclipse 实现 JPA 框架环境

为了提高开发效率，本节将介绍如何通过开发环境 MyEclipse 来开发 JPA 框架方面的应用。MyEclipse 开发环境不仅支持 JPA 框架，而且还可以通过图形界面使 JPA 框架的实现非常简单，如何实现对 JPA 框架的支持？具体步骤如下。

（1）新建一个名 JPA 的 Web Project，详细设置如图 2.63 所示。在 MyEclipse 开发环境中实现该功能非常简单，所以不再讲述。

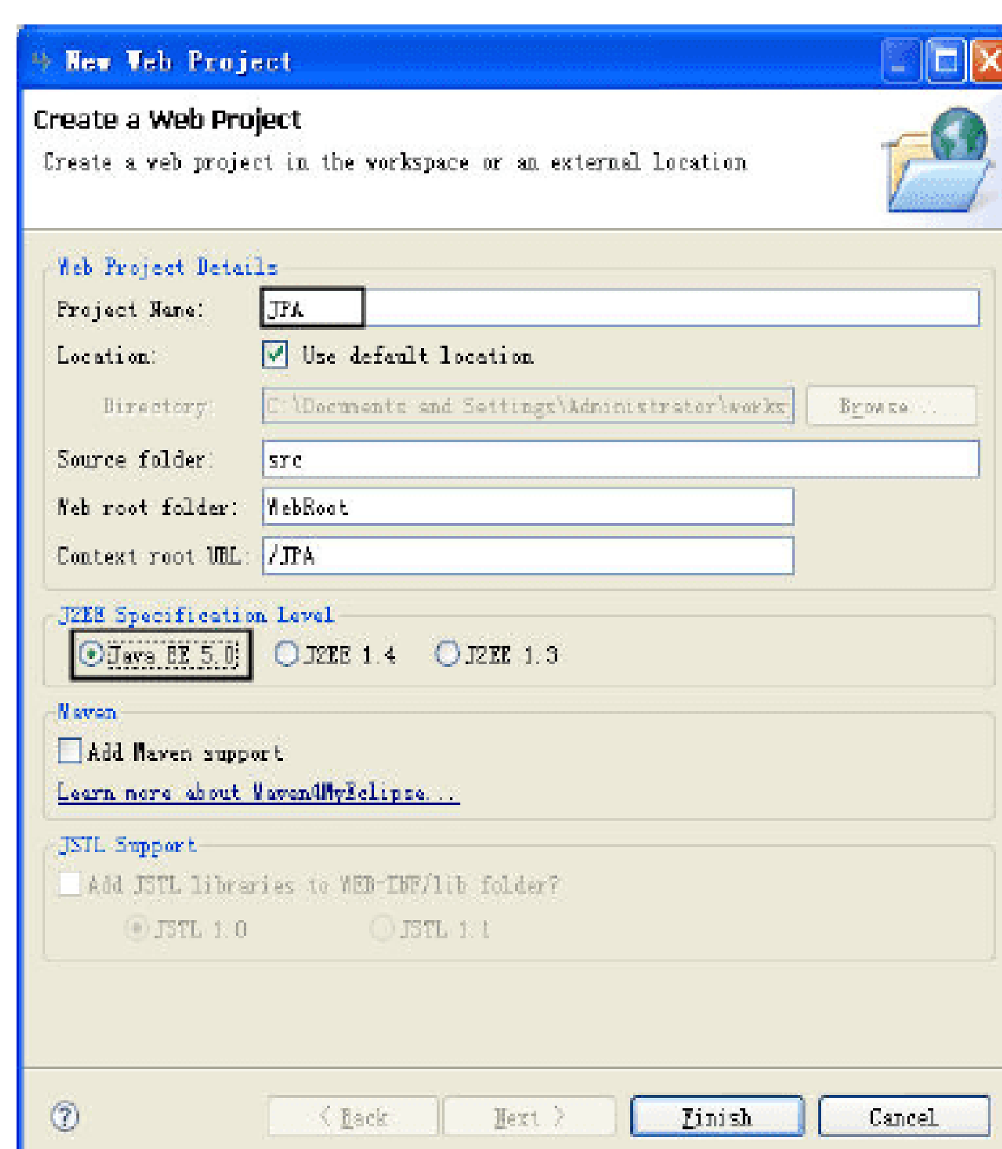


图 2.63　新建 Web Project

（2）为项目增加 JPA 框架的相关类库与文件。在 Package Explorer 视图中右击 JPA 项目，在弹出的快捷菜单中选择 MyEclipse|Project Capabilities|Add JPA Capabilities 命令（如

图 2.64 所示），打开 Add JPA Capabilities（添加 JPA 功能）对话框。

注意：该步骤也可以通过选择 MyEclipse|Project Capabilities|Add JPA Capabilities 命令来实现，如图 2.65 所示。

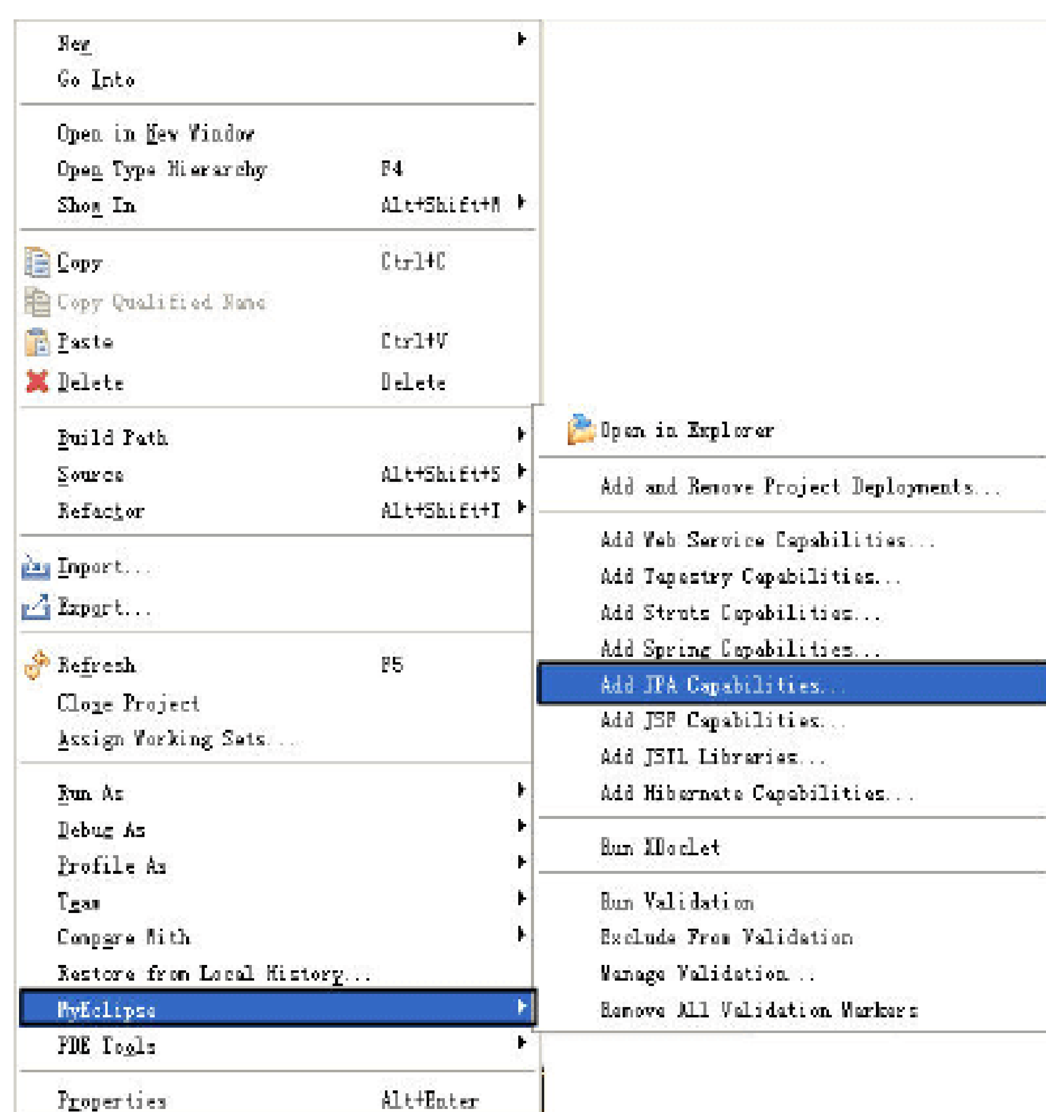


图 2.64　通过右键实现添加 JPA

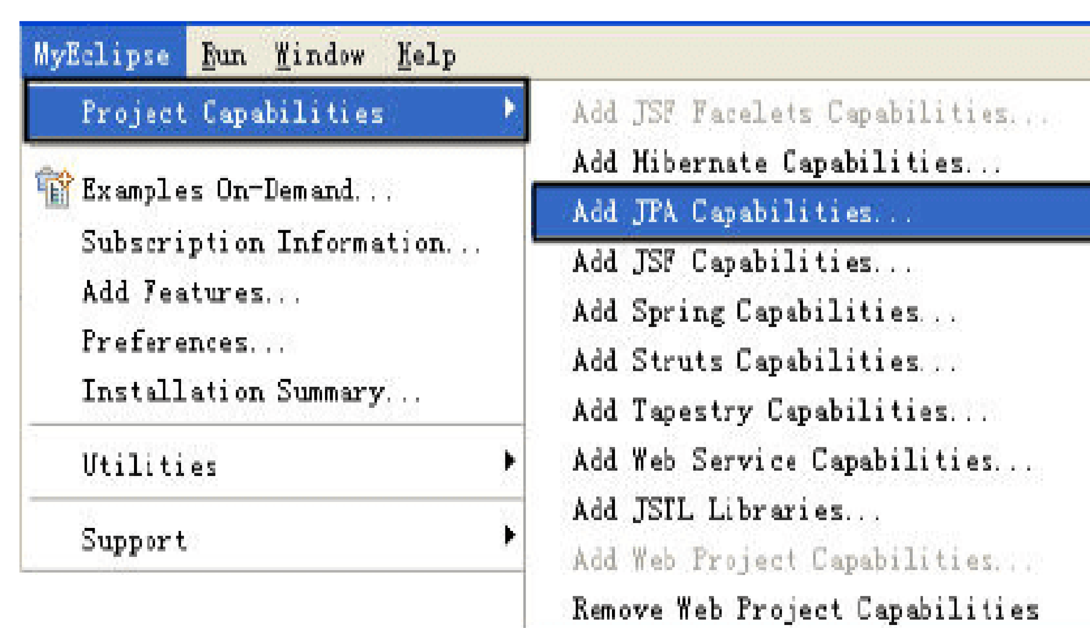


图 2.65　通过菜单实现添加 JPA

在出现的如图 2.66 所示的添加 JPA 功能对话框中，各项一般保持默认值而不需要修改，当然也可以根据需要通过修改一些地方来定制将来生成的类。各个选项的具体含义如下。

- Persistence provider：用来设置配 JPA 框架提供的对象，在这里选择了大文本（Toplink）对象。
- Select the libraries to add to the buildpath：用来设置要加入项目类路径的类库，在这里使用默认选项。

（3）接着单击 Next 按钮，就会弹出显示 Configure Persistence Unit（配置持久化单元名）对话框，如图 2.67 所示。



图 2.66　添加 JPA 框架

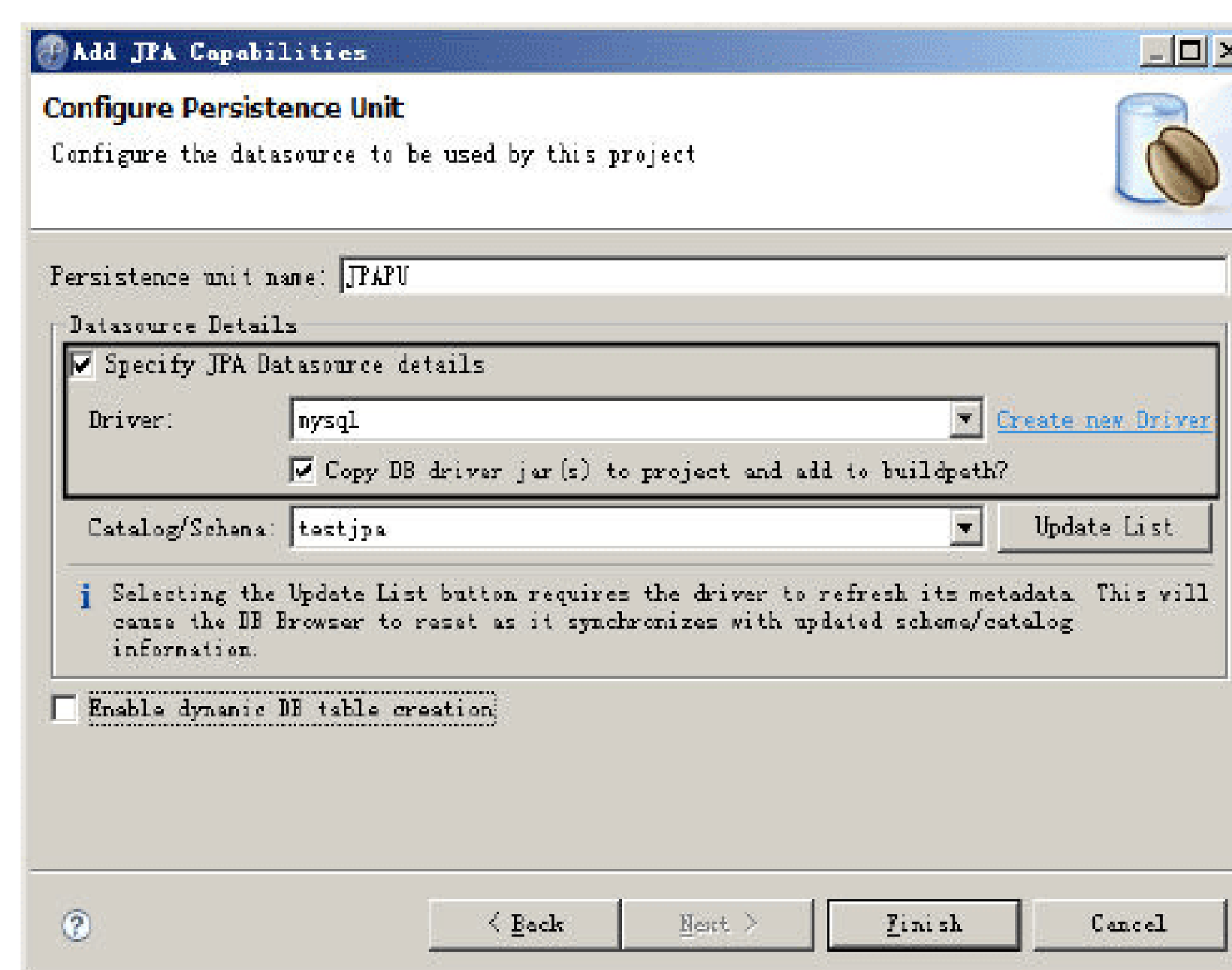


图 2.67　配置持久化单元名

单击 Driver 右侧的现有数据库连接列表，选择创建好的数据库 MySQL，其他相关的连接信息将会自动填入到对话框中。

注意：当选择 Copy DB driver jar(s) to project and to buildpath 复选框后，则会自动添加数据库驱动类库 jar 文件到项目的类路径中。

当完成增加 JPA 相关类库与文件操作后，打开 Package Explorer 视图中的 JPA 项目目录，如图 2.68 所示。

- Toplink Essentials：为项目添加 JPA 框架类库。
- persistence.xml：为项目添加的 JPA 框架配置文件。

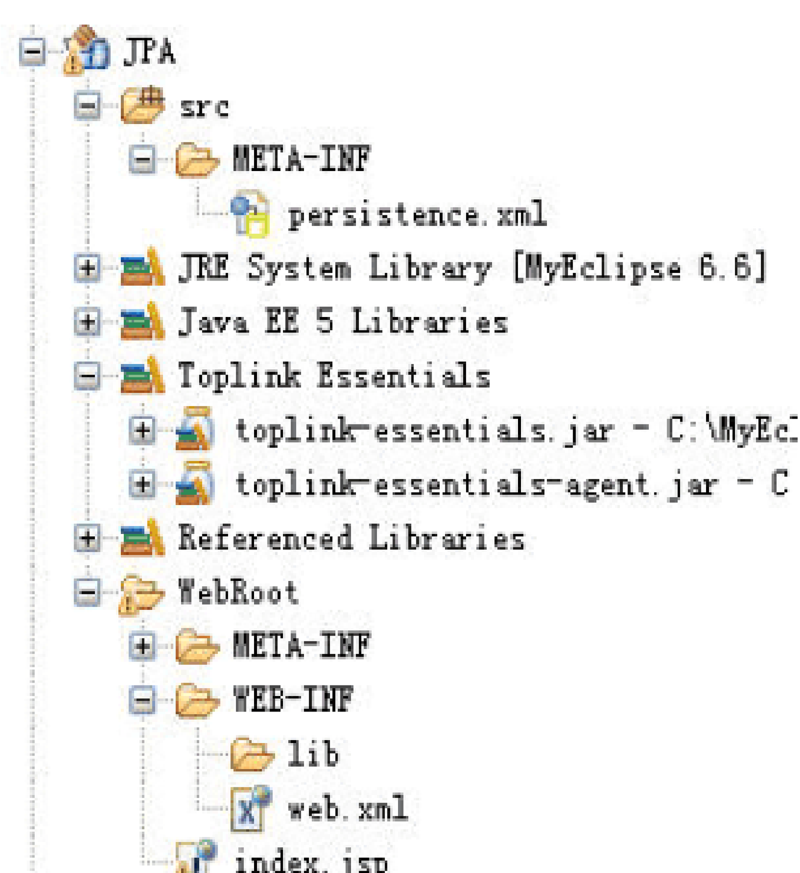


图 2.68　JPA 项目目录

至此，该项目已经具备了 JPA 框架的支持。

2.4.2　MyEclipse 对 JPA 框架支持——添加实体

为了便于讲解，本节将在 2.4.1 小节完成的开发环境中，利用 JPA 框架实现一个具体的功能，即把数据库表转换成实体类。具体过程如下。

在介绍之前，先熟悉一下 JPA 配置文件编辑器，通过双击 persistence.xml 文件，可以打开 JPA 配置文件编辑器，如图 2.69 所示。

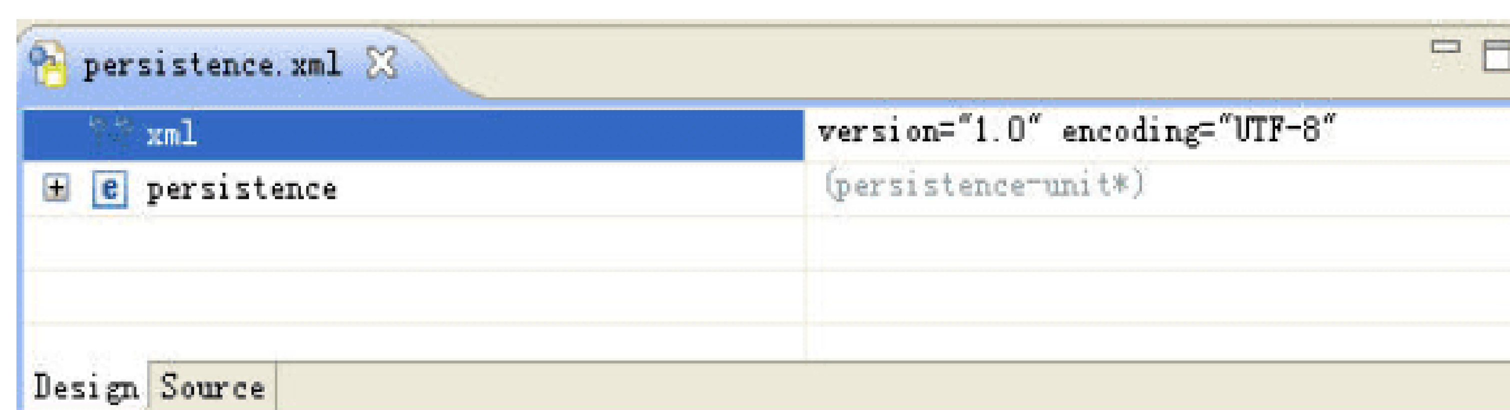


图 2.69　JPA 配置文件编辑器

首先利用 MyElipse 将 JPA 框架的实体类和操作类添加入到 JPA 工程中。该功能的实现与 Hibernate 框架的逆向工程的实现很相似，所以就不详细讲解。首先在 DB Browser 视图中打开 MySQL 数据库连接，然后在该数据库连接中选择要持久化的表格 users。右击该表格，在弹出的快捷菜单中选择 JPA Reverse Engineering 命令（如图 2.70 所示），打开 JPA Reverse Engineering（JPA 逆向工程向导）对话框（如图 2.71 所示）。

各个选项的具体含义如下。

- Java src folder：POJO 和 DAO 生成后源代码的文件夹。
- Java package：POJO 和 DAO 生成后所在包。
- Entity Bean Generation：设置实体 Bean，如果选择了 Create abstract class 选项，则会创建出抽象类；如果选择了另一个选项，则会在创建实体类的同时将实体类用<class>元素加入配置文件。在该项目中选择了第二项。

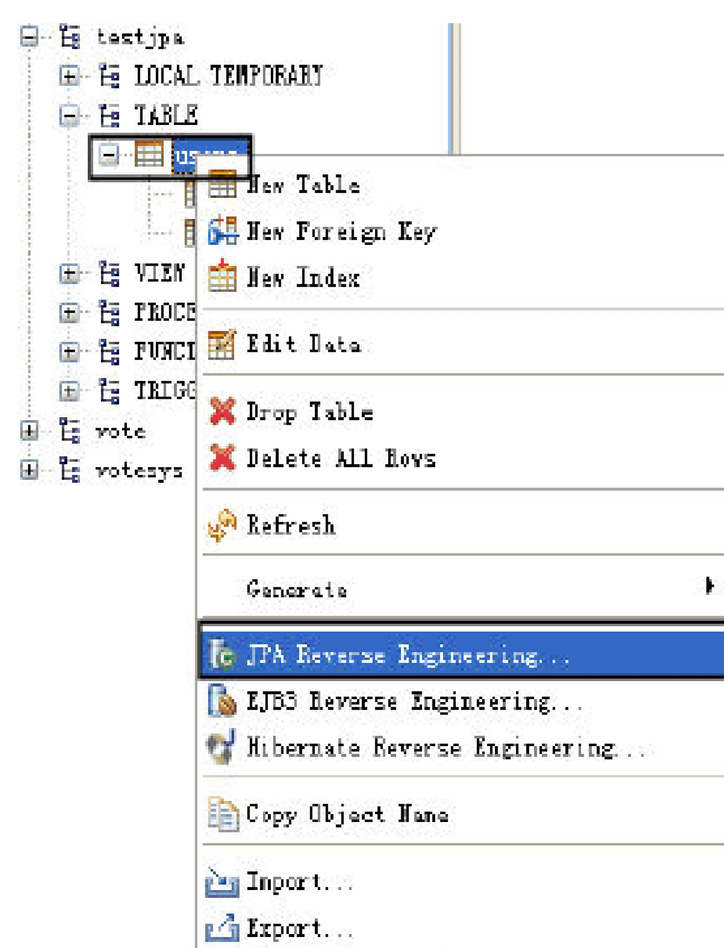


图 2.70　逆向工程菜单

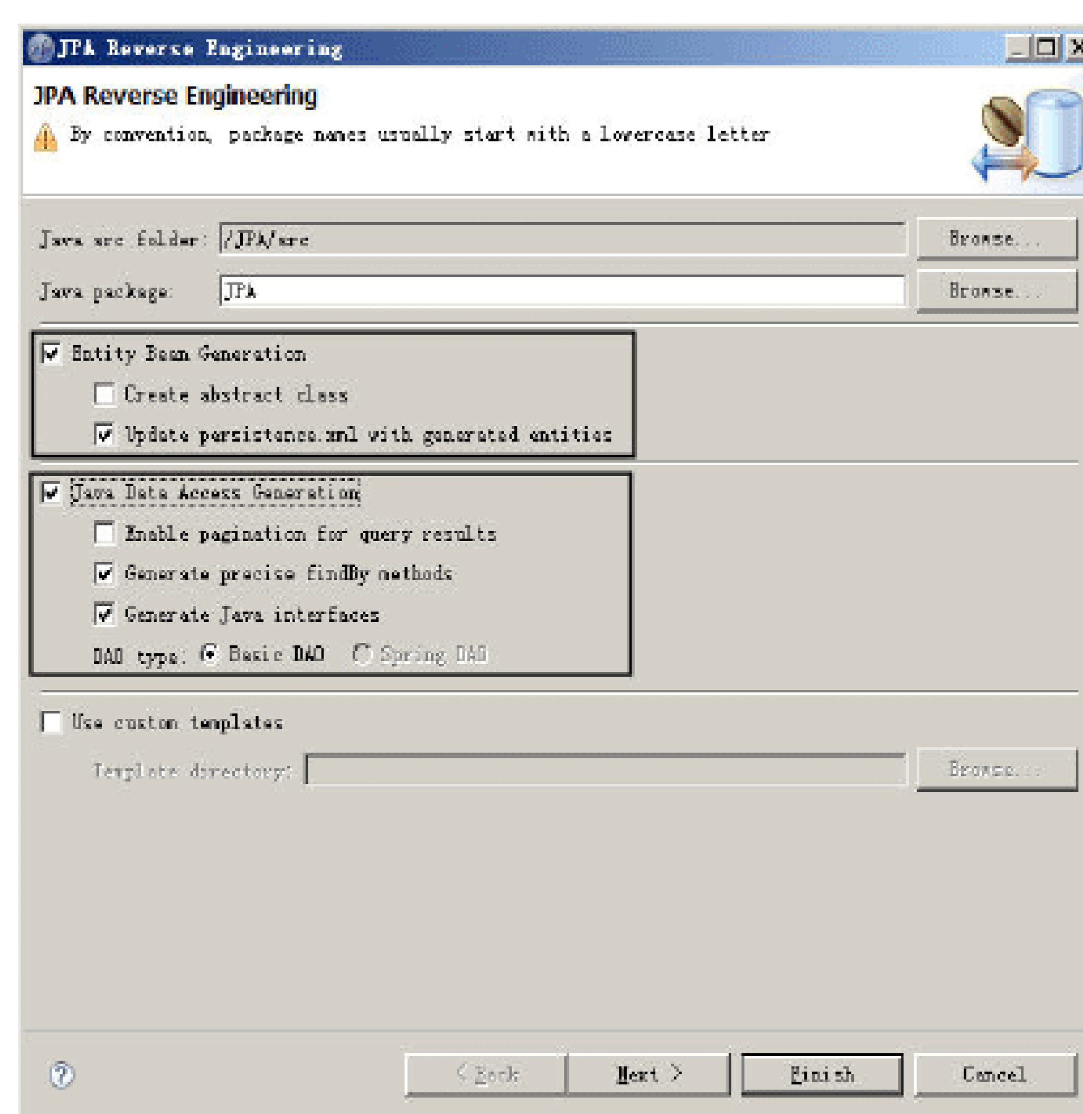


图 2.71　逆向工程

- ❑ Java Data Access Generation：设置 Java 数据存取方式是否采用分页查询，如果选择了第一项，则会启用分页查询；如果选择了第二项，则会创建精确查找方法；如果选择了第三项，则会创建操作数据库的接口。在该项目中选择了第二项和 Basic DAO 接口。
- ❑ Use custom templates：用来设置是否选择自定义模板。

注意：虽然 MyEclipse 5.5 对 JPA 框架的逆向功能提供帮助，但是在操作类的删除方法中并没有将要删除的变量实体化，所以一定要手工修改该错误。MyEclipse6 已经修正该错误。

当完成逆向工程的操作后，打开 Package Explorer 视图中的 JAP 项目目录，如图 2.72 所示。

在自动生成的文件中，只需要注意 4 个文件即可。

- ❑ EntityManagerHelper.java：该类实现实体管理器的一些操作。
- ❑ IUsersDAO.java：操作实体类的接口。
- ❑ UsersDAO.java：该类是接口类的实现类。
- ❑ Users.java：关于数据库表的实体类。

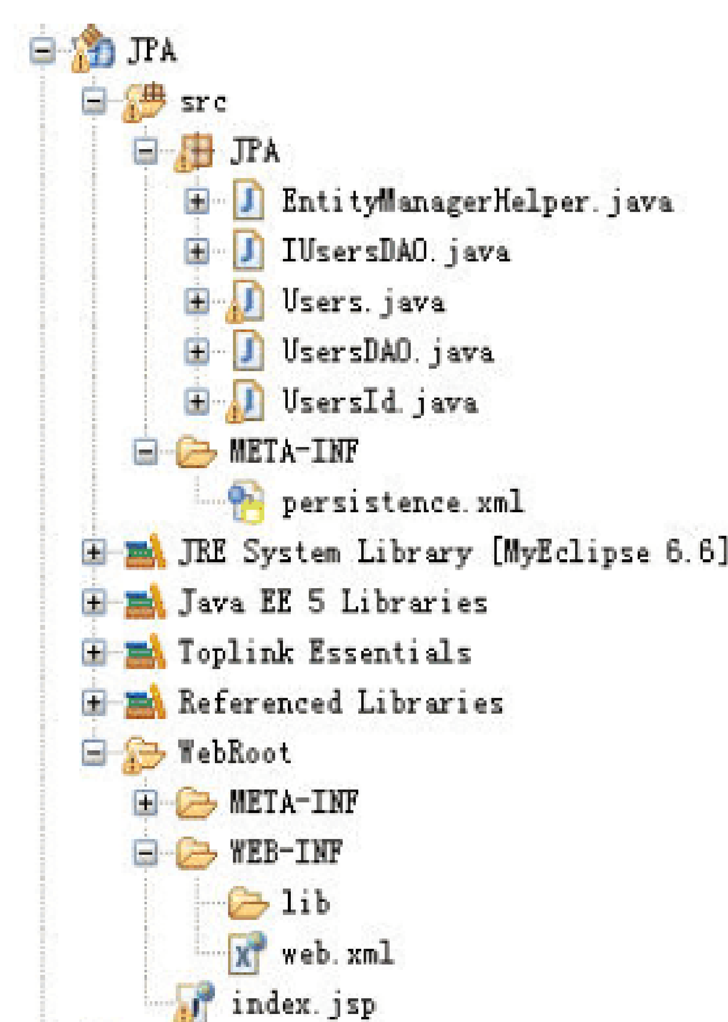


图 2.72　JPA 项目目录

2.4.3　MyEclipse 对 JPA 框架支持——单个类转成 JPA 实体

为了便于讲解，本节将接着 2.4.2 节完成的项目环境中继续讲解，利用 MyEclipse 开发工具中的 JPA Details 面板将一个测试类转成实体类。即在 src 文件夹下再添加一个实体类，具体步骤如下。

（1）首先在包 testclass 中创建一个名为 Numbe 的 JavaBean 测试类，具体内容如代码 2.17 所示。

代码 2.17　测试类：Numbe.java

```
...
```

```
public class Numbe {
    //创建属性
    private String name;
    private int num;
    //省略相关属性的 get()和 set()方法
...
}
```

（2）关于类名的配置，首先将鼠标放在类的名称 Numbe 上，这时 JPA Details 面板的 Map As 选项上就会出现一个下拉列表框，然后选择其中的 Entity 选项（如图 2.73 所示），该面板就会变成如图 2.74 所示。JPA Details 面板中 General 选项卡上各项的意义如下。

- Table：用来设置类名，默认为 Numbe。
- Name：用来设置类对应表名，默认为 Numbe。
- Attribute Overrides：用来设置属性重写。

当选项卡为 Inheritance 时，用来设置实体类的继承关系，如图 2.75 所示。

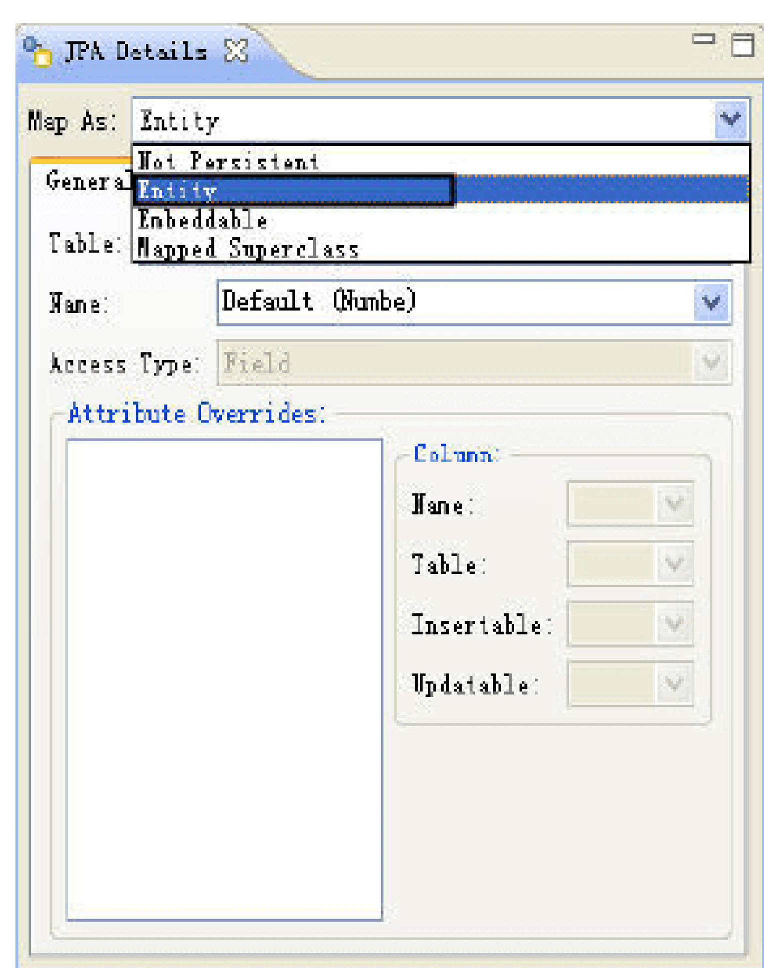


图 2.73　选择 Entity 选项

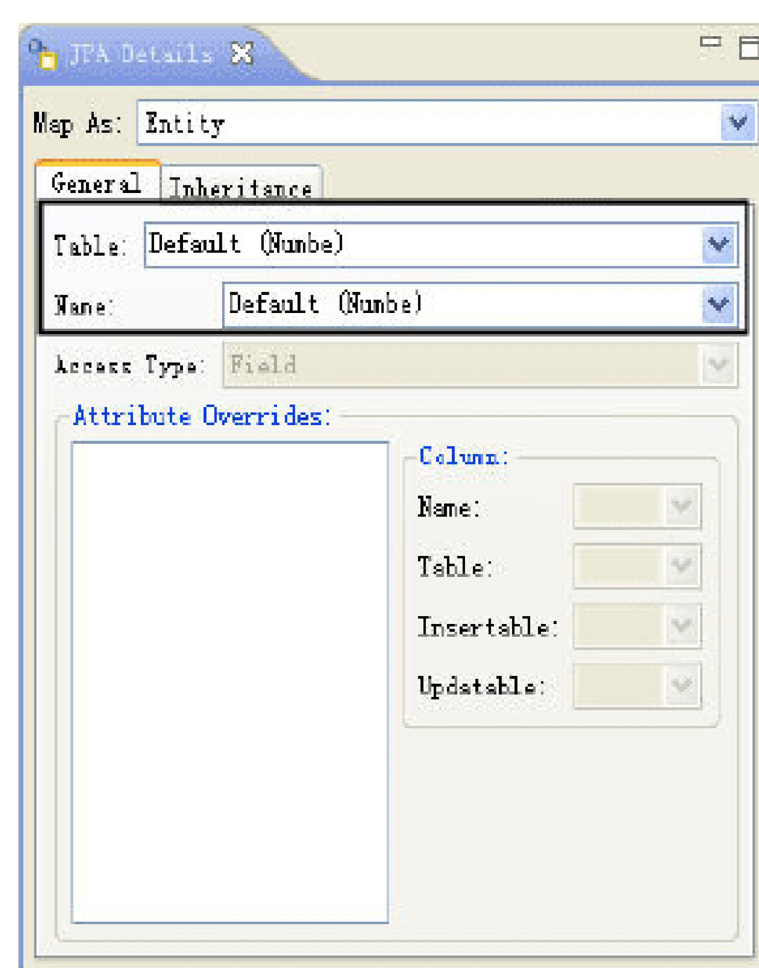


图 2.74　关于 Entity 面板

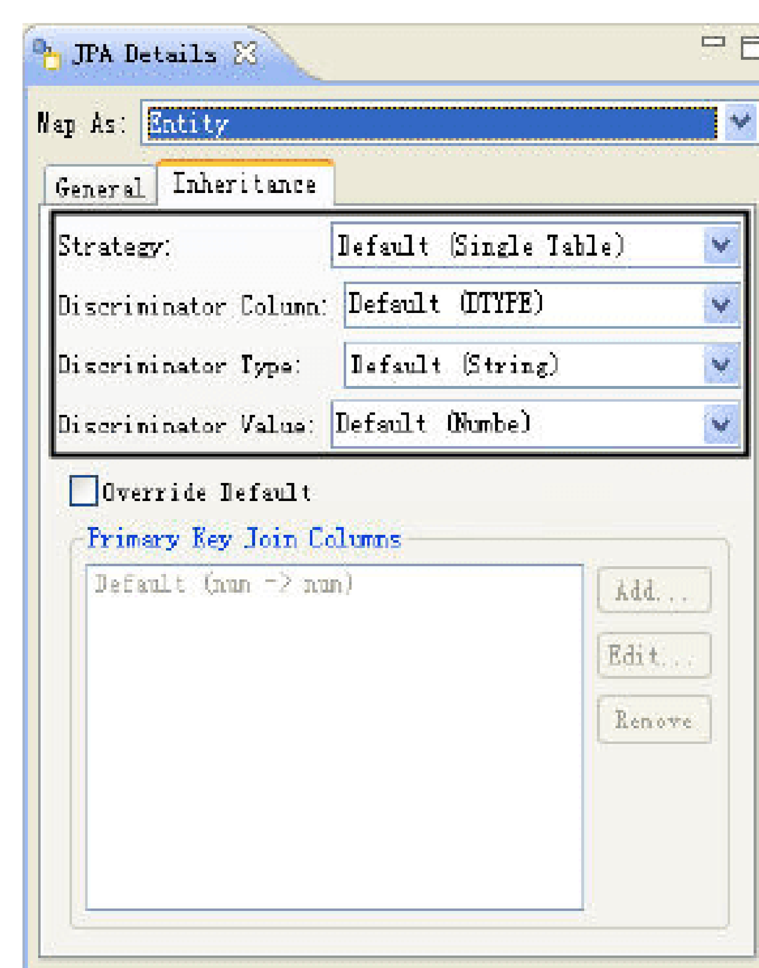


图 2.75　Inheritance 选项卡

该选项卡各项的意义如下。

- Strategy：用来设置继承策略。
- Discriminator Column：用来设置标识列。
- Discriminator Type：用来设置标识列的类型。
- Discriminator Value：用来设置标识列中对应这个类的值。
- Override Default：用来设置属性重写。

上述各项均为默认值。

（3）接着设置类中的变量，首先设置 name 变量，将鼠标放在变量 name 上，这时 JPA Details 面板就会变成如图 2.76 所示的操作字段的面板。

JPA Details 面板中 General 选项卡各项的意义如下。

- Map As：用来设置映射方式。
- Name：用来设置属性对应字段，默认为 name。
- Table：用来设置属性对应字段所在数据表，默认为 Numbe。
- Insertable：用来设置是否可以新建，默认值就可以。
- Updatable：用来设置是否可以修改，默认值就可以。

当选项卡为“PK Generation”时（如图 2.77 所示），各项的意义如下。

- ❑ Primary Key Generation：用来设置主键产生的策略。
- ❑ Table Generator：用来设置主外键。
- ❑ Sequence Generator：用来设置主键顺序。

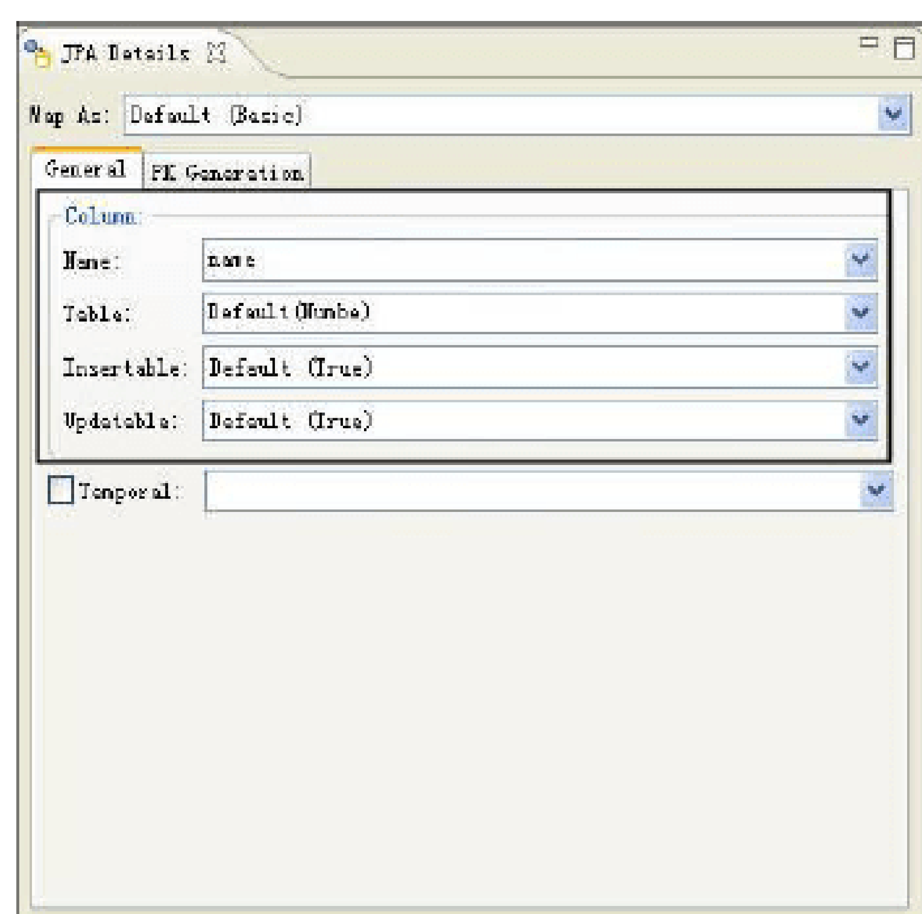


图 2.76　设置变量 name 面板

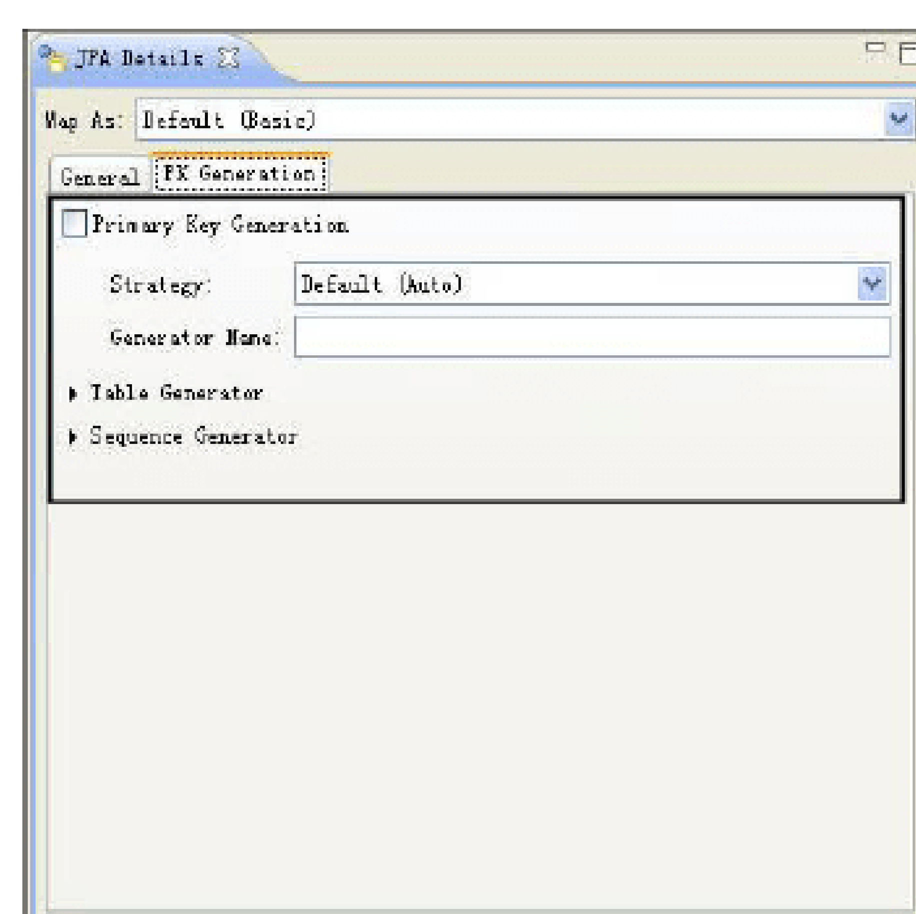


图 2.77　PK Generation 选项卡

上述各项均为默认值。

（4）然后设置 num 变量，将鼠标放在变量 num 上，这时 JPA Details 面板就会变成如图 2.78 所示的操作字段的面板。由于该属性要成为主键，所以需要在如图 2.79 所示的选项卡上进行相应的设置。

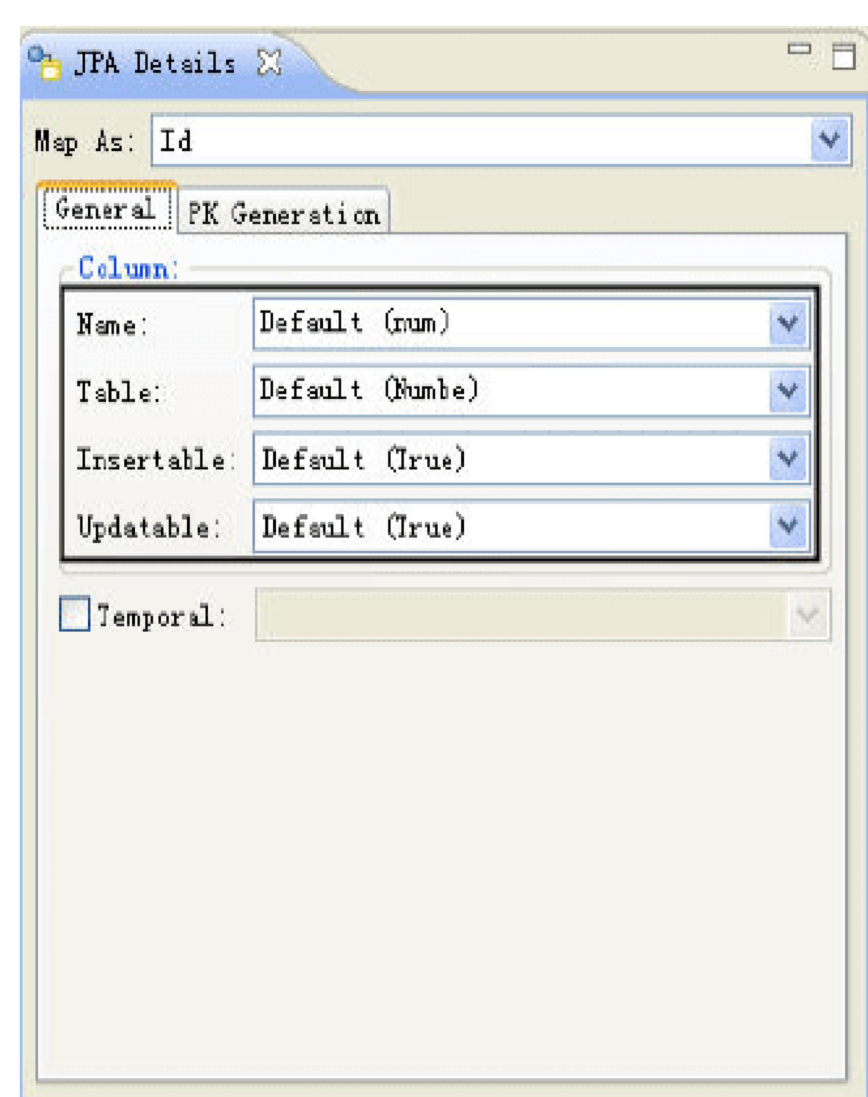


图 2.78　设置变量 num 面板

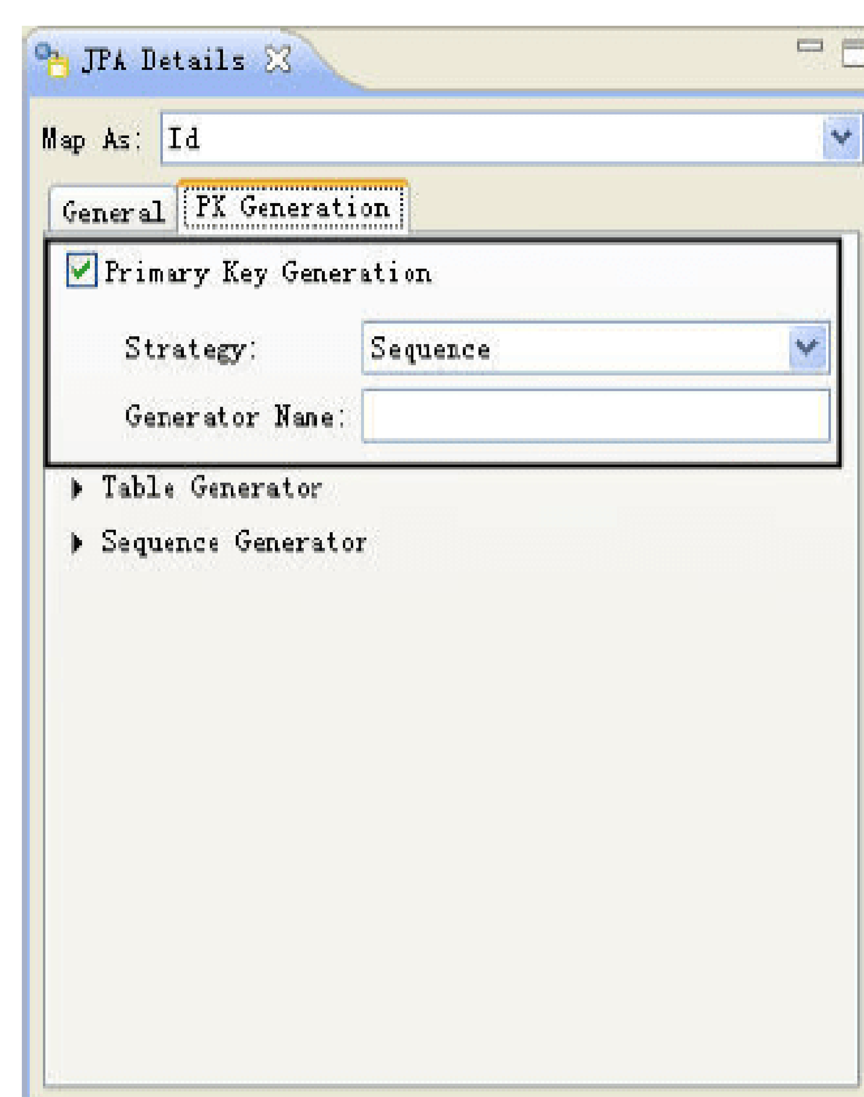


图 2.79　PK Generation 选项卡

（5）通过对上述步骤的设置后，Numbe 类的具体内容就变成如代码 2.18 所示。最后，还需要对该类在 persistence.xml 文件中进行配置，具体内容如代码 2.19 所示，这时 Numbe 类才是真正的实体类。

代码 2.18　实体类：Numbe.java

```
...
@Entity
@Table(name = "Numbe", catalog = "testjpa")
public class Numbe {
    @Column(name = "name")
    private String name;                                  //标注属性 name
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int num;                                              //标注属性 num
    //配置相关属性的 get()和 set()方法
...
}
```

代码 2.19　实体类：persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence 1 0.xsd" version=
   "1.0">

    <persistence-unit name="JPAPU" transaction-type="RESOURCE LOCAL">
        <provider>                      <!--配置管理器-->
            oracle.toplink.essentials.PersistenceProvider
        </provider>
        <class> testclass.Numbe</class>      <!--配置实体类 Numbe-->
        <class> JPA.UsersId</class>
        <properties>
            <!--配置数据库驱动-->
            <property name="toplink.jdbc.driver"
                value="com.mysql.jdbc.Driver" />
            <!--配置数据库连接 URL-->
            <property name="toplink.jdbc.url"
                value="jdbc:mysql://localhost:3306/testjpa" />
            <!--配置数据库连接的用户名-->
            <property name="toplink.jdbc.user" value="root" />
            <!--配置数据库连接的密码-->
            <property name="toplink.jdbc.password" value="root" />
        </properties>
    </persistence-unit.>
</persistence>
```

至此，就完成实体类的转换。

2.5　Spring 框架的实现

为了让读者熟练地掌握基于 Spring 框架的应用，本节没有对 Spring 的 IOC 和 AOP 等概念做详细的讲解，而是具体讲解了如何使用开发环境 MyEclipse 来实现 Spring 框架应用的开发。

2.5.1　用 MyEclipse 实现 Spring 框架环境

为了提高开发效率，本节将介绍如何通过开发环境 MyEclipse 来开发 Spring 框架方面的应用。MyEclipse 开发环境实现了对 Spring 框架的全方位支持，如何实现对 Spring 框架的支持呢？具体步骤如下。

（1）新建一个名 Spring 的 Web Project，详细设置如图 2.80 所示。在 MyEclipse 开发环

境中实现该功能非常简单，所以不再讲述。

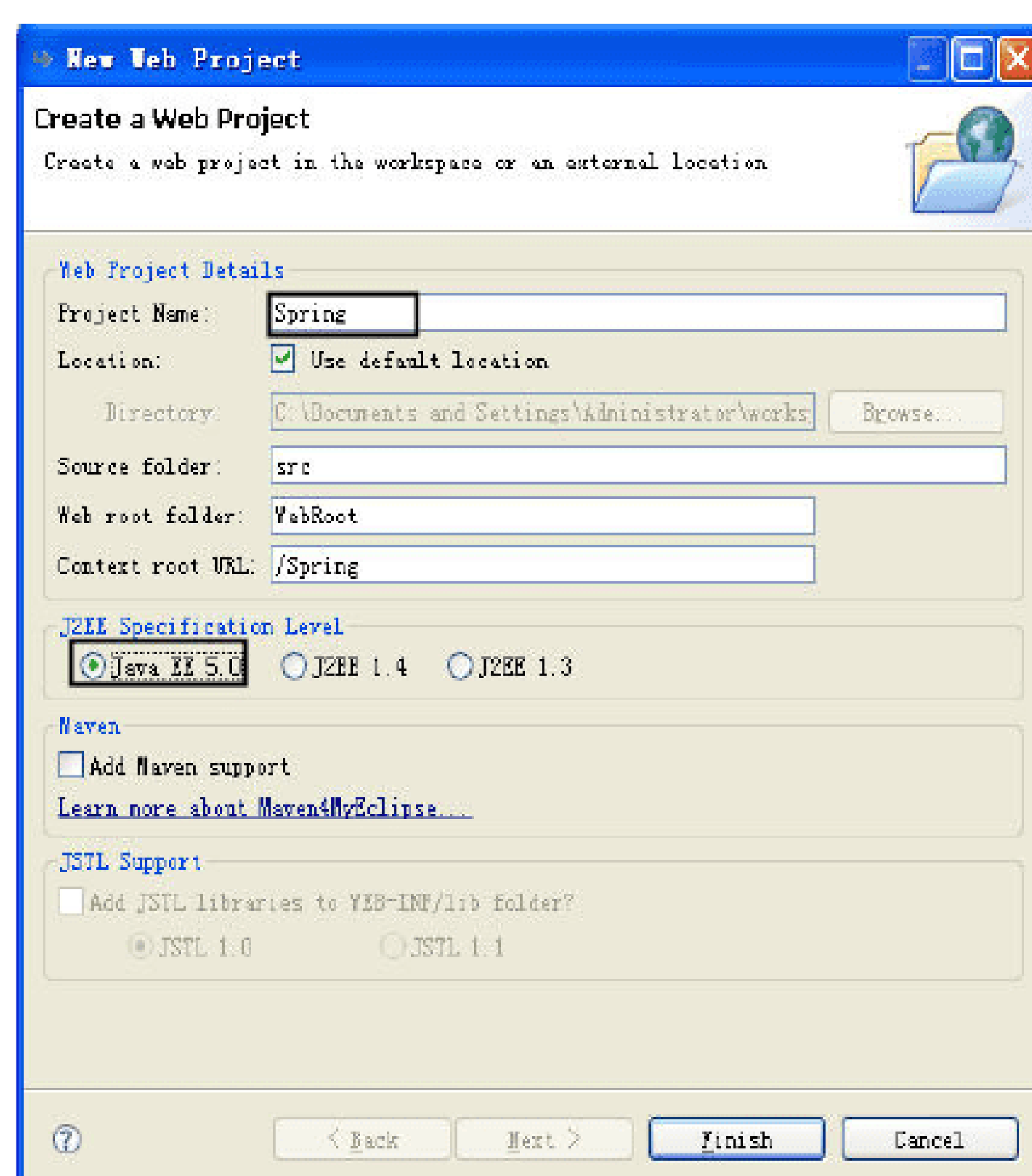


图 2.80　新建 Web Project

（2）为项目增加 Spring 相关类库与文件。在 Package Explorer 视图中右击该项目，在弹出的快捷菜单中选择 MyEclipse|Project Capabilities|Add Spring Capabilities 命令，如图 2.81 所示来实现。

注意：该步骤也可以通过选择 MyEclipse|Project Capabilities|Add Spring Capabilities 命令来实现，如图 2.82 所示。

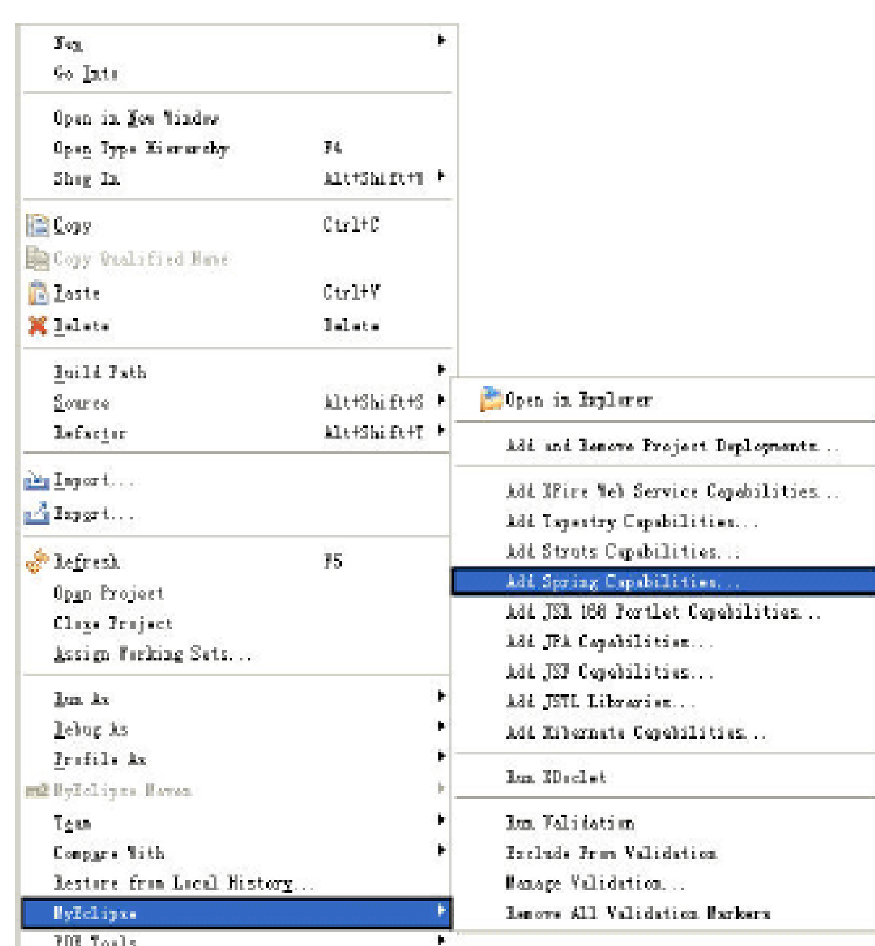


图 2.81　通过右击实现添加 Spring

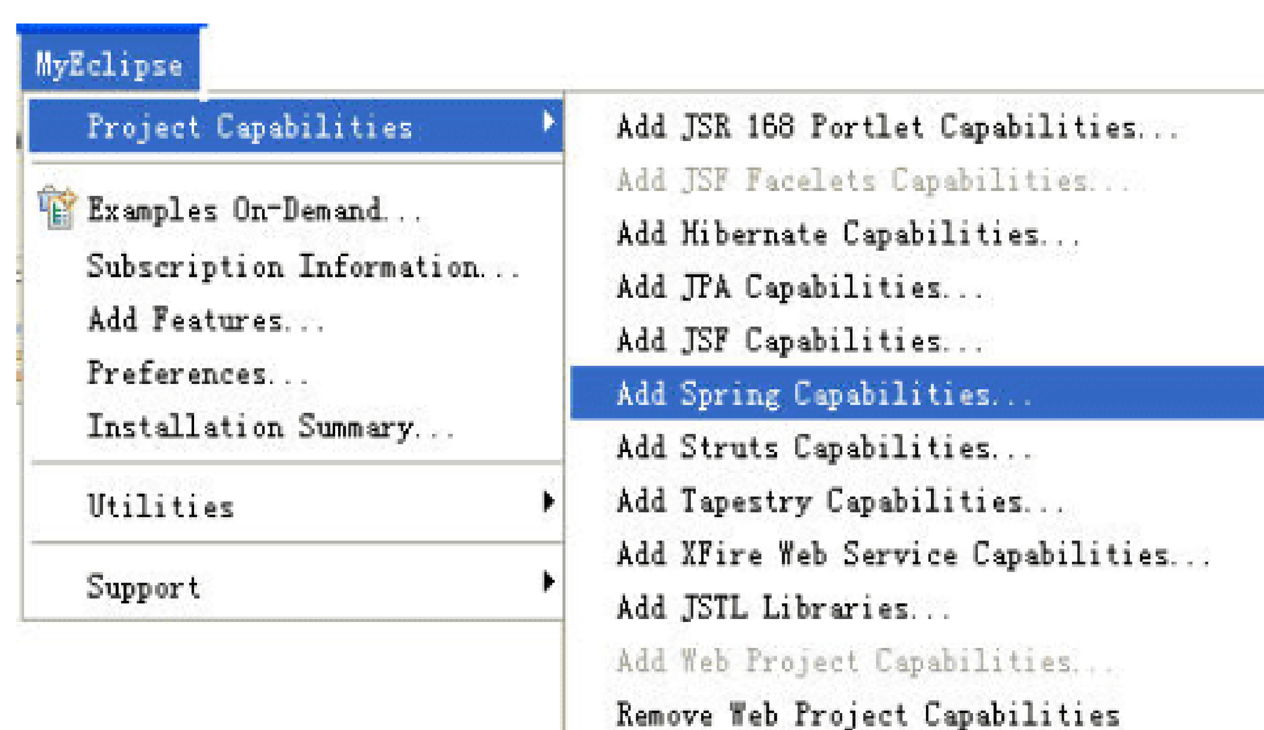


图 2.82　通过菜单实现添加 Spring

在弹出的图 2.83 所示的 Add Spring Capablilities（添加 Spring 功能）对话框中，默认值一般不需要修改就可以使用，当然也可以根据需要通过修改一些地方来定制将来生成的类。各个选项的具体意义如下。

- Spring version：用来设置配 Spring 框架的版本，该选项一般选择 Spring 2.0。
- Select the libraries to add to the buildpath：用来设置要加入项目类路径的类库。

注意：Spring 框架的类库是按照模块进行划分的，所以可以根据项目的需要选择必要的类库，而不需要选择全部的类库。

- JAR Library Installation：指定 JAR 类库的按照方式。上面的单选按钮表示把引用的类库加入类路径；下面的单选按钮则是把指定一个目录的所有 JAR 文件和元素

文件加入到当前项目。

- ❑ Tag Library Installation：用来指定元素库文件的安装目录。

当完成增加 Spring 相关类库与文件操作后，展开 Package Explorer 视图中 Spring 项目目录，如图 2.84 所示。

- ❑ Spring 2.0 Core Libraries：为项目添加 Spring 核心类库。
- ❑ applicationContext.xml：为项目添加的 Spring 配置文件。

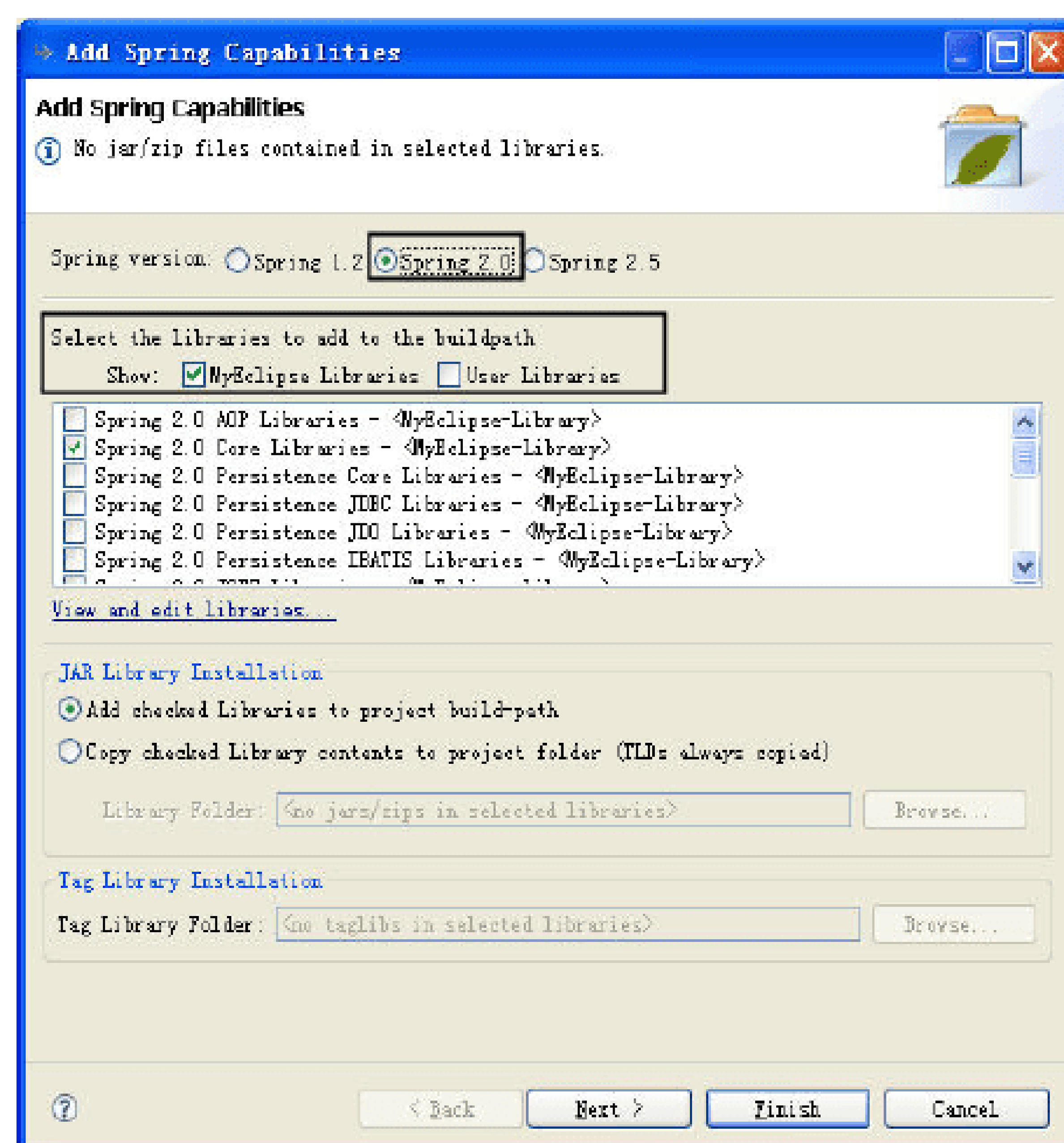


图 2.83　添加 Spring 框架

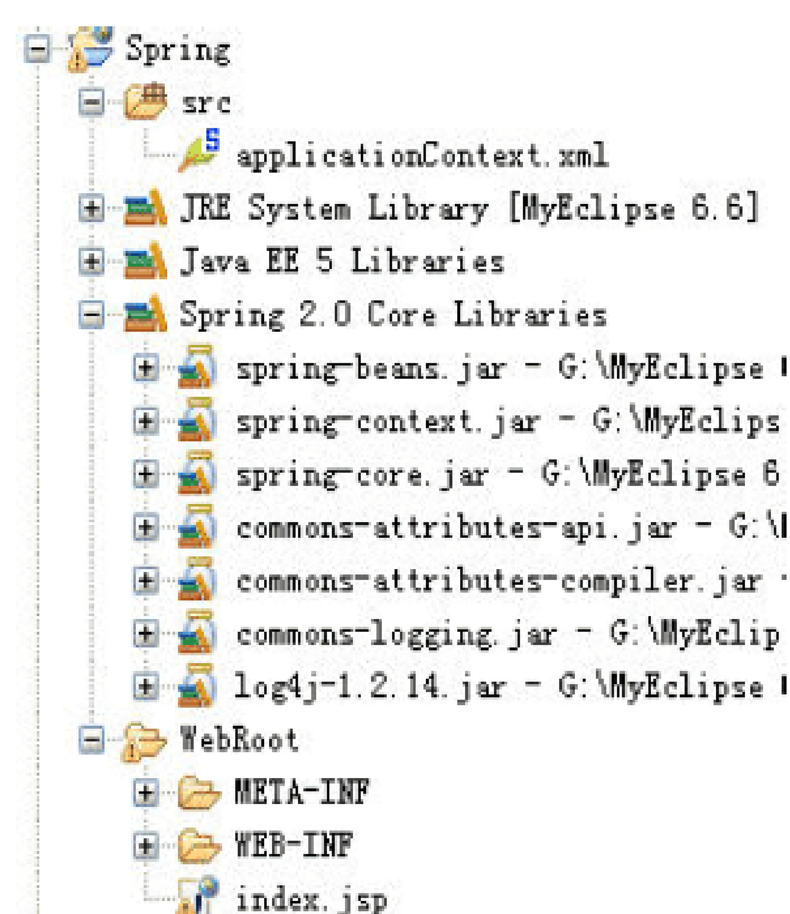


图 2.84　Spring 项目目录

至此，该项目已经具备了 Spring 框架的支持。

2.5.2　用 MyEclipse 实现 Spring 项目

为了便于讲解，本节将在 2.5.1 节完成的开发环境中，利用 Spring 框架实现一个具体的应用。首先介绍代码的运行背景，利用 Spring 的反射方式调用 JavaBean 的 get()和 set()方法。具体步骤如下。

在介绍之前，先熟悉一下 Spring 配置文件编辑器，通过双击 applicationContext.xml 文件，可以打开 applicationContext.xml 配置文件编辑器，如图 2.85 所示。

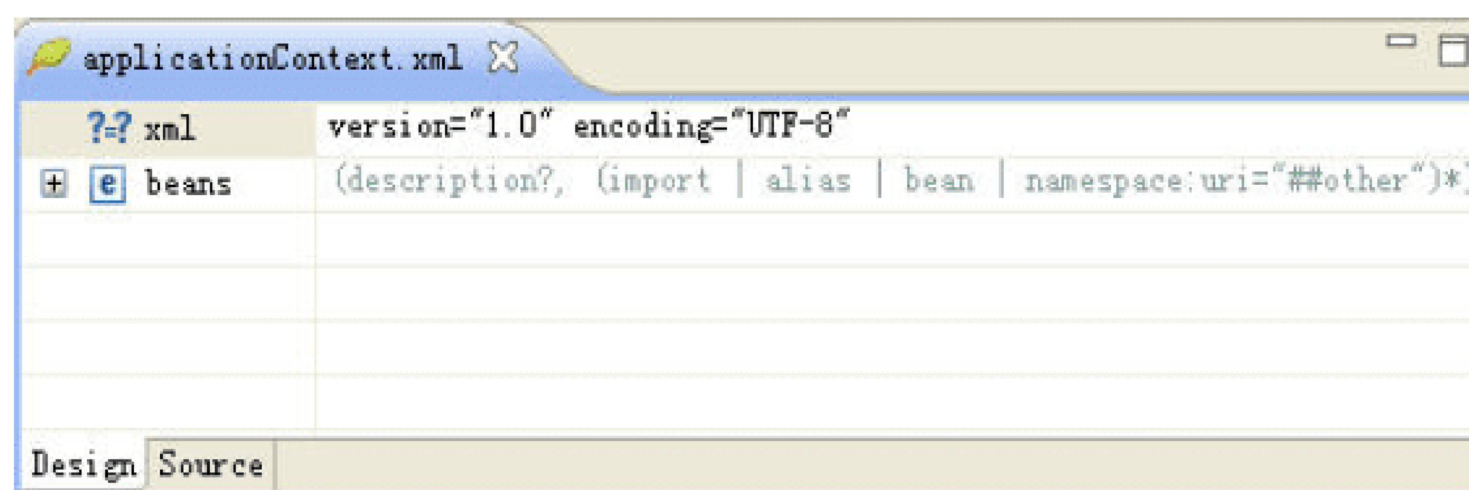


图 2.85　applicationContext.xml 配置文件编辑器

注意：applicationContext.xml 文件的文件名可以改成其他名字。

通过右击画布面板，在弹出的快捷菜单中选择 Spring 选项，就会出现 5 个选项（如图 2.86 所示）来实现 Spring 项目的组件，前 4 个选项的具体含义如下。

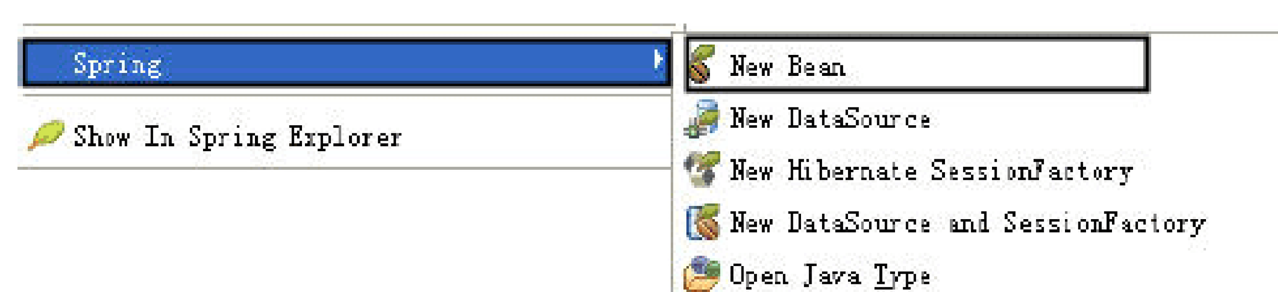


图 2.86　Spring 项目的组件

- New Bean：新建 Bean。
- New DataSource：新建数据。
- New Hibernate SessionFactory：新建 Hibernate 会话工厂。
- New DataSource and SessionFactory：新建数据源和会话工厂。

① 在目录 Spring/src 中创建一个名为 Message 的 JavaBean，代码 2.20 用来作为测试 Spring 反射反射的 Bean。

代码 2.20　简单的 Bean：Message.java

```
...
public class Message {
    private String content;                                //创建属性
    //省略属性 content 的 get()和 set()方法
...
}
```

② 为 Message.java 创建配置信息，MyEclipse 开发工具对其提供了全方位的支持。在图 2.86 中选择 New Bean 选项，就会弹出如图 2.87 所示的 New Spring Bean（创建 Spring Bean）对话框。

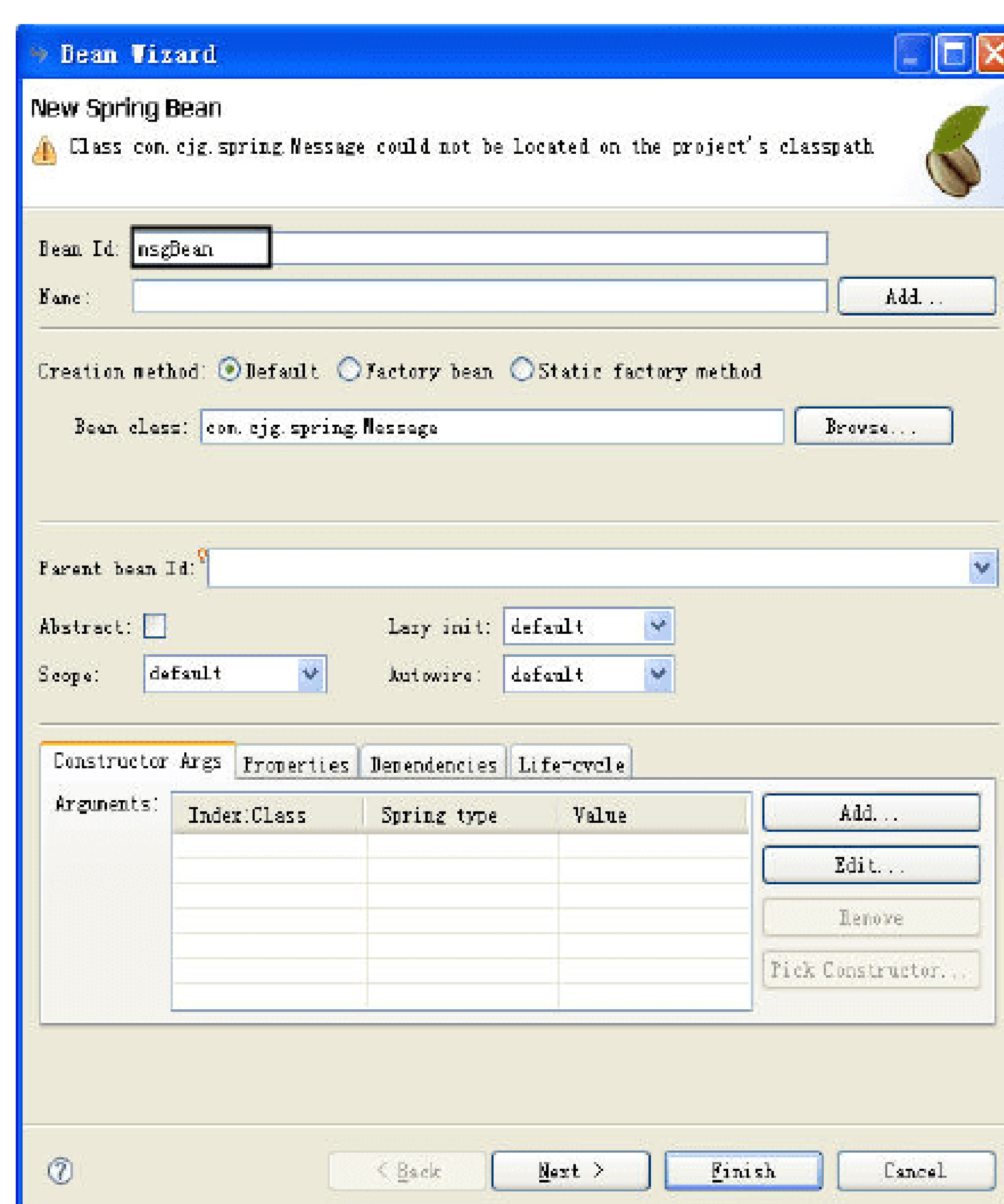


图 2.87　配置 Spring Bean

在 Bean Id 文本框中输入 msgBean，然后在 Bean class 选项中选择编写好的关于 JavaBean 的完整路径。而其他选项一般不需要用，保持默认就可以。当完成上述操作后，打开 applicationContext.xml 文件，就会比没有配置前的内容多出如下的代码。

```
<bean id="msgBean" class="com.cjg.spring.Message">
</bean>
```

当该项目中其他文件想使用 Message 对象时，可以通过 msgBean 对象来调用。msgBean 对象由 Spring 框架的核心容器来创建和管理。

③ 在目录 Spring/src 中创建一个名为 SpringTest 的 java 类，代码 2.21 通过 Spring 的控制反转调用测试 Bean 的方法。

代码 2.21　调用 Bean 的方法：SpringTest.java

```
...
public class SpringTest {
    public static void main(String[] args) {
    ClassPathXmlApplicationContext ctx = new ClassPathXmlApplication
    Context("/applicationContext.xml");               //读取配置文件
    Message msg=(Message)ctx.getBean("msgBean");      //获取 JavaBean 的对象
    msg.setContent("Spring 信息");                    //设置属性的值
    System.out.println(msg.getContent());             //输出属性的值
    }
}
```

④ 运行 SpringTest.java 文件，运行结果如图 2.88 所示。

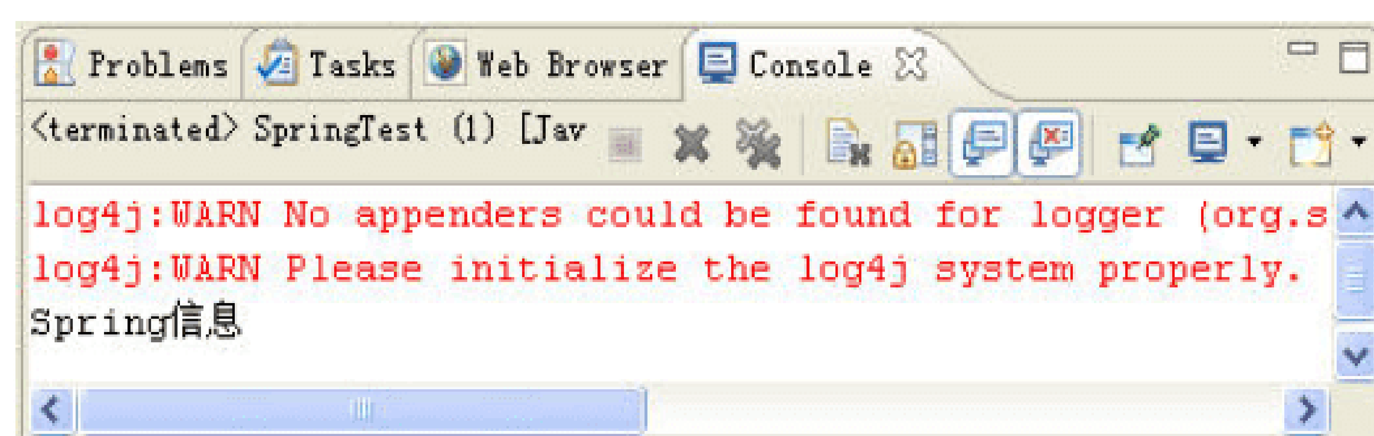


图 2.88　运行结果

2.5.3　MyEclipse 对 Spring 框架方面的支持

在 Spring 框架中可以利用其提供的 JdbcTemplate 类和 DataSource 来开发基于 JDBC 的应用。那么如何实现 DataSource 方面的配置？MyEclipse 开发工具对 DataSource 的配置做了全方位的支持。

首先在实现基于 JDBC 的应用时，除了必要的 Spring 核心类库外，还必须有 Spring Persistence JDBC Libraries 类库。在 MyEclipse 开发工具中如何实现对 DataSource 方面的配置？

右击 Spring 配置文件，在弹出的快捷菜单中选择 Spring|New DataSource 命令，弹出如图 2.89 所示的 New Spring DataSource（创建 DataSource 配置）对话框。在该对话框中首先在 Bean Id 文本框中输入数据源 Bean 的 id（符合名字规格就可以），然后在 DB Driver 下拉列表框中选择创建好的数据库连接就可以，这些数据库连接则是在 Database Explorer 视图中建立的。只需要对该两项进行填写，其他项就会自动填写。打开 Spring 的配置文件 applicationContext.xml，其中就会多出如代码 2.22 所示的内容。

代码 2.22　配置文件：applicationContext.xml

```
...
        <!--设置数据源对象-->
```

```
        <bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
        <!--设置数据源驱动-->
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
        </property>
        <!--设置数据源 URL-->
        <property name="url" value="jdbc:mysql://localhost/mysql">
        </property>
        <property name="username" value="root"></property><!--设置用户名-->
        <property name="password" value="root"></property><!--设置密码-->
...
```

在开发与 Hibernate 框架集成的项目时，肯定会用到 SessionFactory 对象，如何在 MyEclipse 开发环境创建和配置 SessionFactory 对象呢？

右击 Spring 配置文件，在弹出的快捷菜单中选择 Spring|New DataSource and SessionFactory 命令，弹出如图 2.90 所示的 New Spring Hibernate SessionFactory（创建 SessionFactory 配置）对话框。

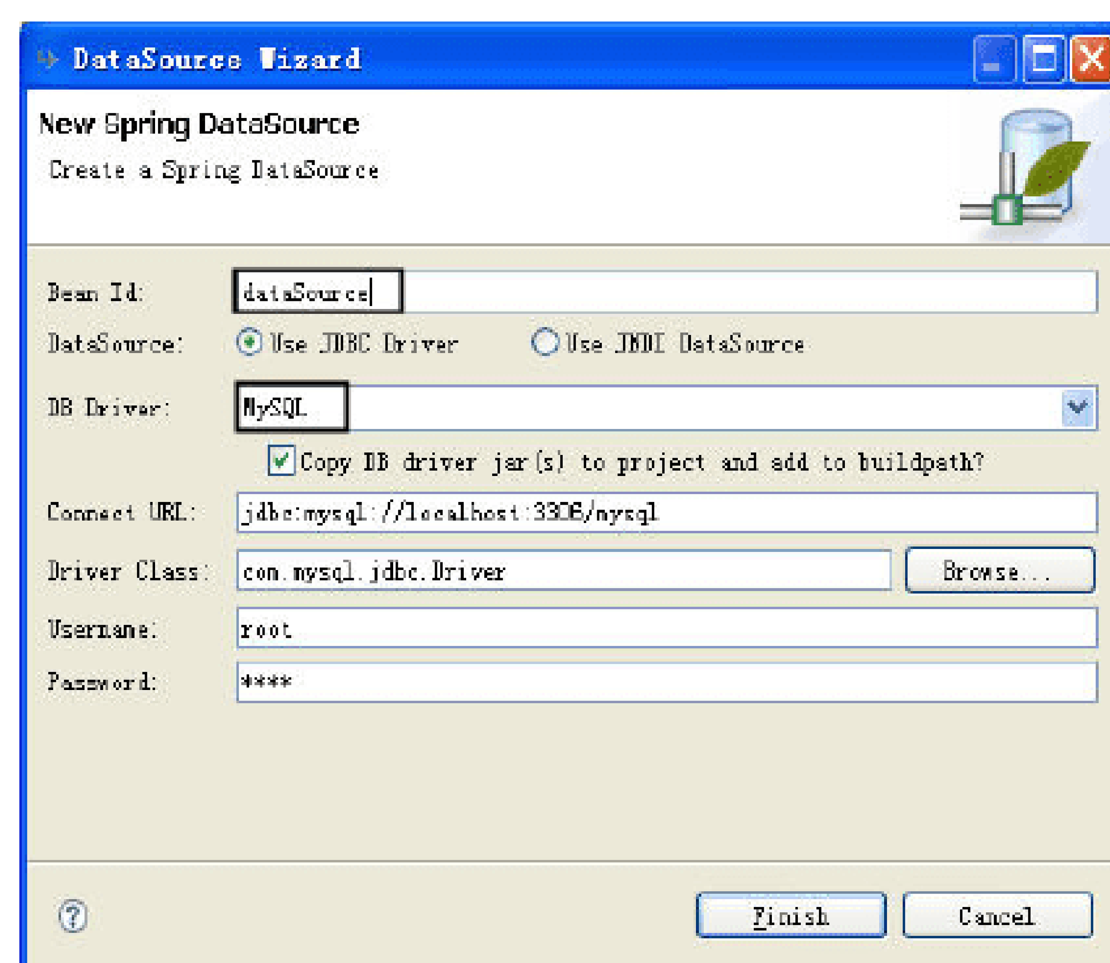


图 2.89　DataSource 配置对话框

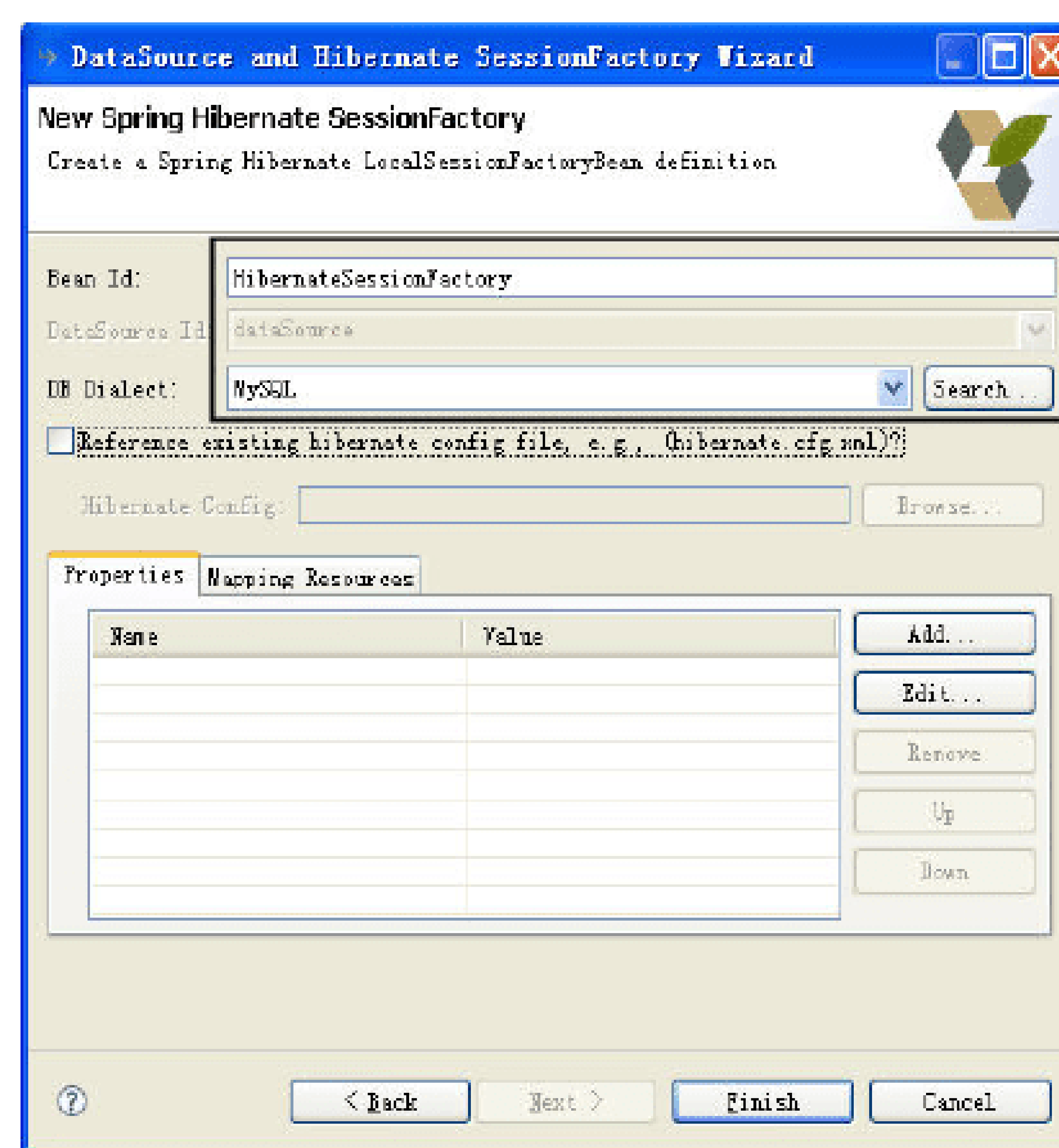


图 2.90　SessionFactory 配置对话框

在该对话框中可以通过定义 DataSource 然后配置映射资源的方式创建一个 HibernateSessionFactory。首先在对话框的 Bean Id 中填写 springHibernateSessionFactory，然后在 DataSource 中选择配好的 DataSource，最后为 Hibernate Config 选择配置文件的路径。打开 Spring 的配置文件 applicationContext.xml，其内容就会多出如代码 2.23 所示的内容。

代码 2.23　配置文件：applicationContext.xml

```
...
    <!--设置 SessionFactory 对象-->
    <bean id="springHibernateSessionFactory"
        class="org.springframework.orm.hibernate3.LocalSession
        FactoryBean">
        <!--设置数据源对象-->
        <property name="dataSource">
            <ref bean="dataSource" />
        </property>
        <property name="hibernateProperties">
```

```
            <props>
                <prop key="hibernate.dialect">       <!--配置数据库方言-->
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
...
```

这样就会自动创建好关于 Hibernate SessionFactory 对象的配置文件。

2.6　JSF 框架的实现

为了让读者熟练地掌握基于 JSF 框架的应用，本节没有对 JSF 的相关概念做详细讲解，而是具体讲解了如何使用开发环境 MyEclipse 来实现 JSF 框架应用的开发。

2.6.1　用 MyEclipse 实现 JSF 框架环境

为了提高开发效率，本节将介绍如何通过开发环境 MyEclipse 来开发 JSF 框架方面的应用。MyEclipse 开发环境不仅支持前面介绍的几种框架，而且还支持 JSF 框架，如何实现对 JSF 框架的支持呢？具体步骤如下。

（1）新建一个名为 JSF 的 Web Project，详细设置如图 2.91 所示。在 MyEclipse 开发环境中实现该功能非常简单，所以不再讲述。

（2）为项目增加 JSF 相关类库与文件。在 ackage Explorer 视图中右击该项目，在弹出的快捷菜单中选择 MyEclipse|Project Capabilities|Add JSF Capabilities 命令（如图 2.92 所示）来实现。

注意：*该步骤也可以通过菜单 MyEclipse|Project Capabilities|Add JSF Capabilities 来实现，如图 2.93 所示。*

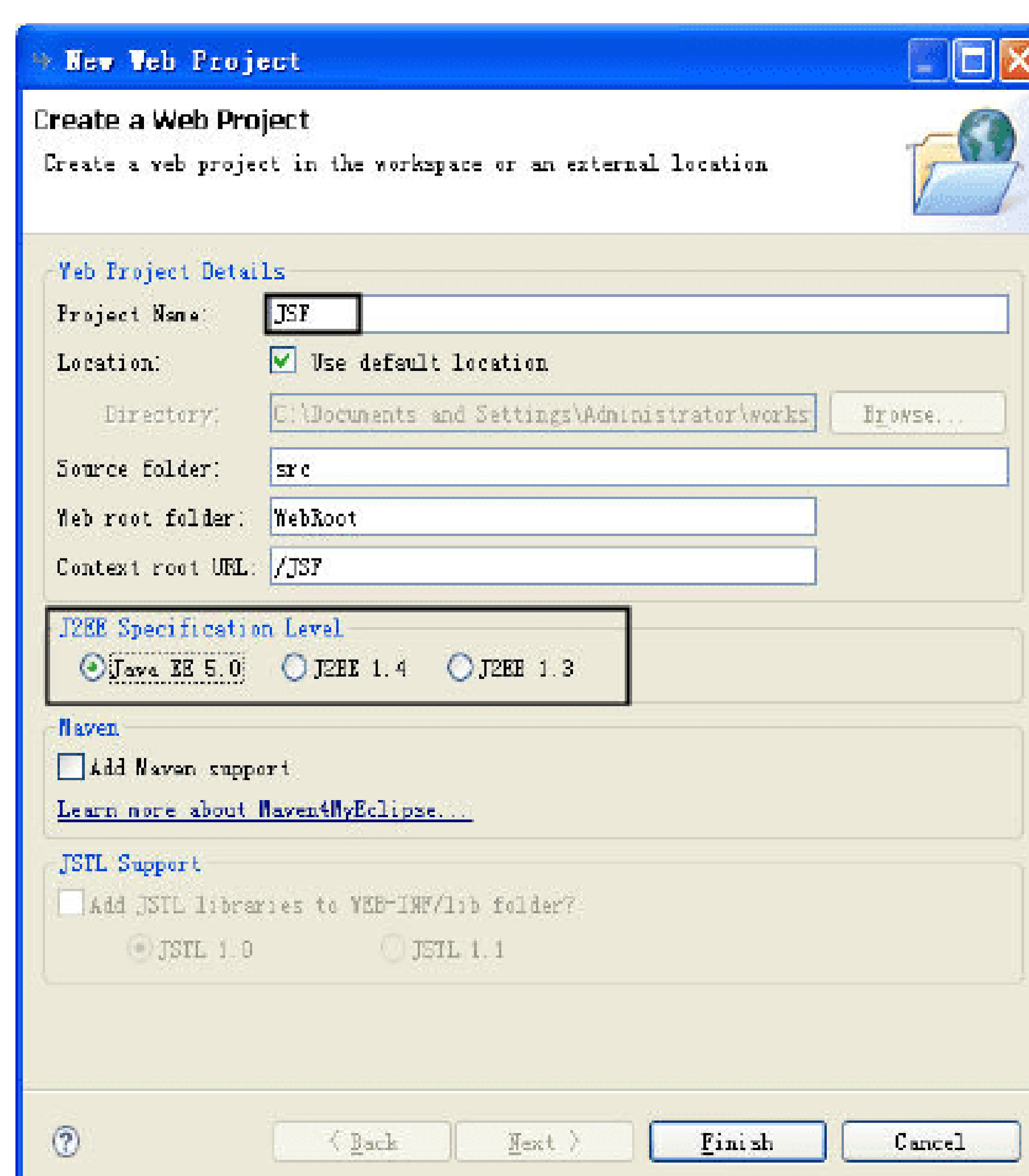


图 2.91　新建 Web Project

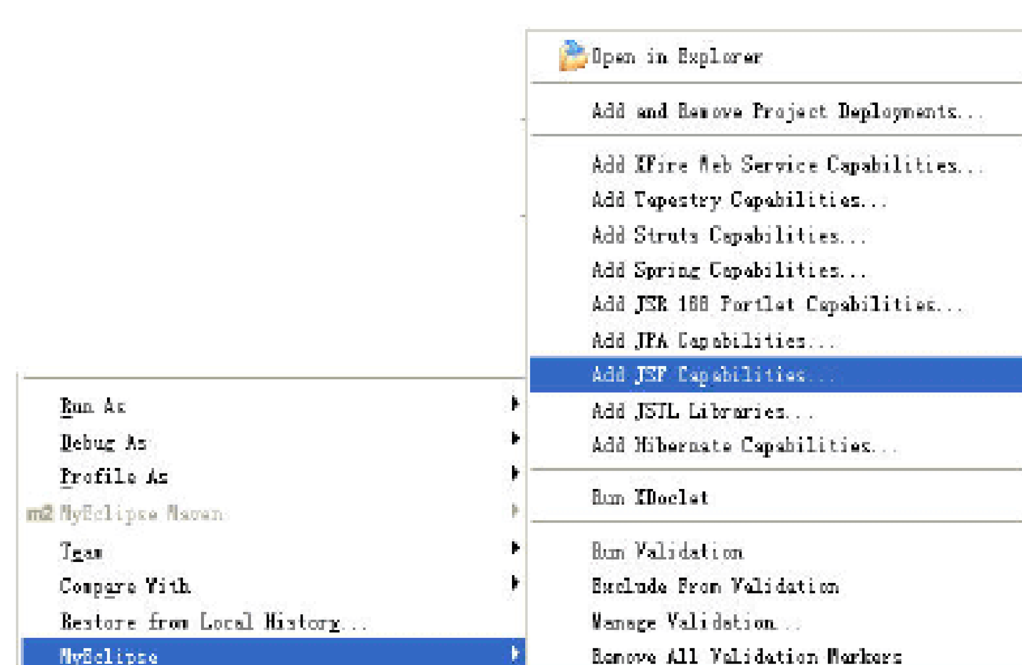


图 2.92　通过右击实现添加 JSF

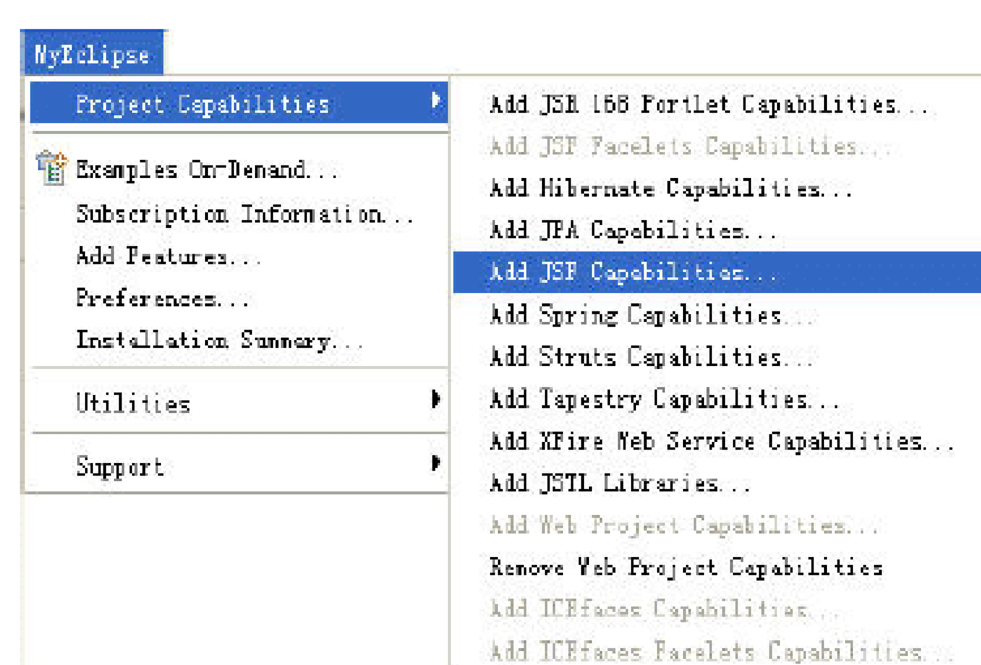


图 2.93　通过菜单实现添加 JSF

在弹出的图 2.94 所示的 JavaServer Faces Support for MyEclipse Web project（添加 JSF 功能）对话框中，默认值一般不需要修改就可以使用，当然也可以根据需要通过修改一些

地方来定制将来生成的类。各个选项的具体含义如下。

- ❑ JSF config path：用来设置 JSF 框架配置文件的位置。
- ❑ Faces servlet name：JSF 框架的核心 Servlet 名字。
- ❑ URL pattern：JSF Servlet 的映射方式，即 JSF Servlet 默认监听的 URL 类型的名为 *.faces。

接着单击 Next 按钮，就会进入 JavaServer Facelets Support（设置 Facelets）对话框，如图 2.95 所示。单击 Finish 按钮就会结束向导。

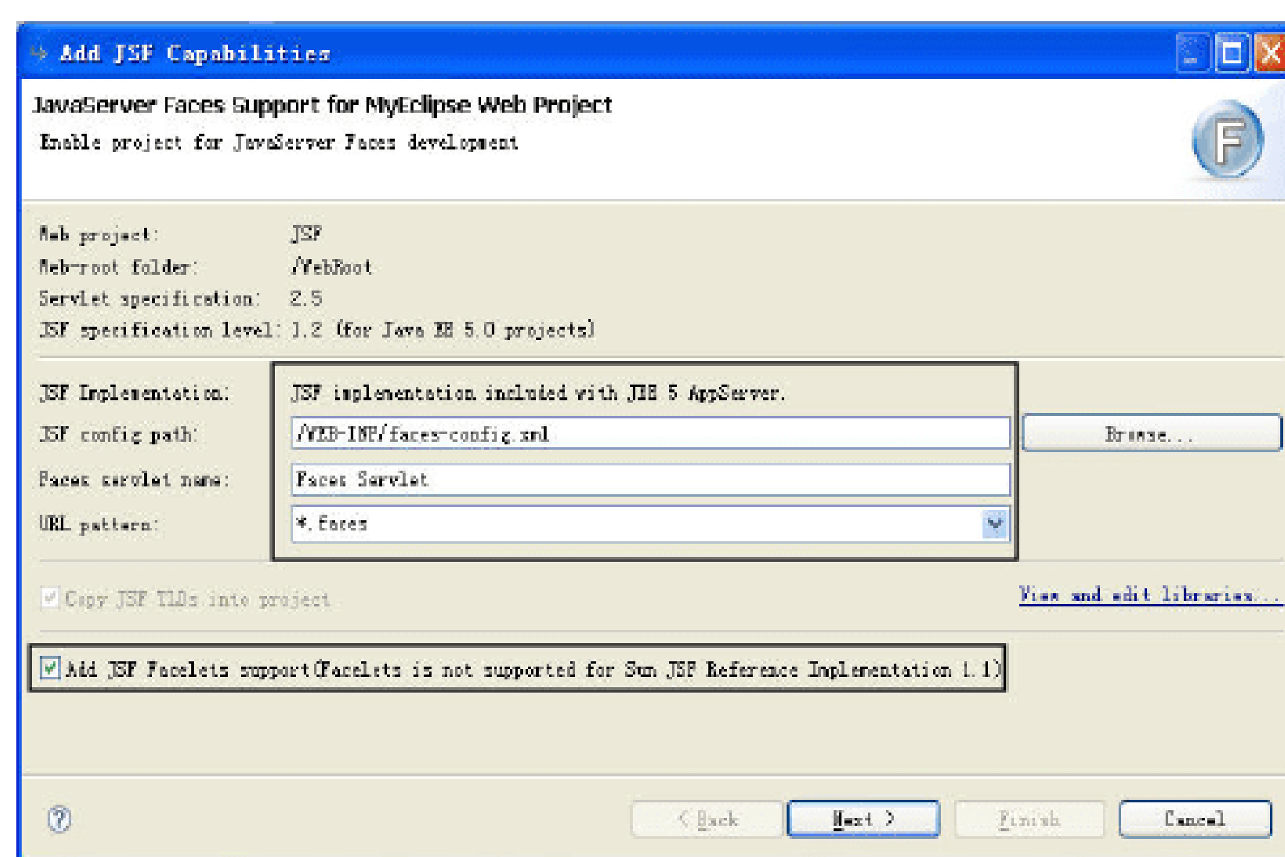


图 2.94　添加 JSF 框架

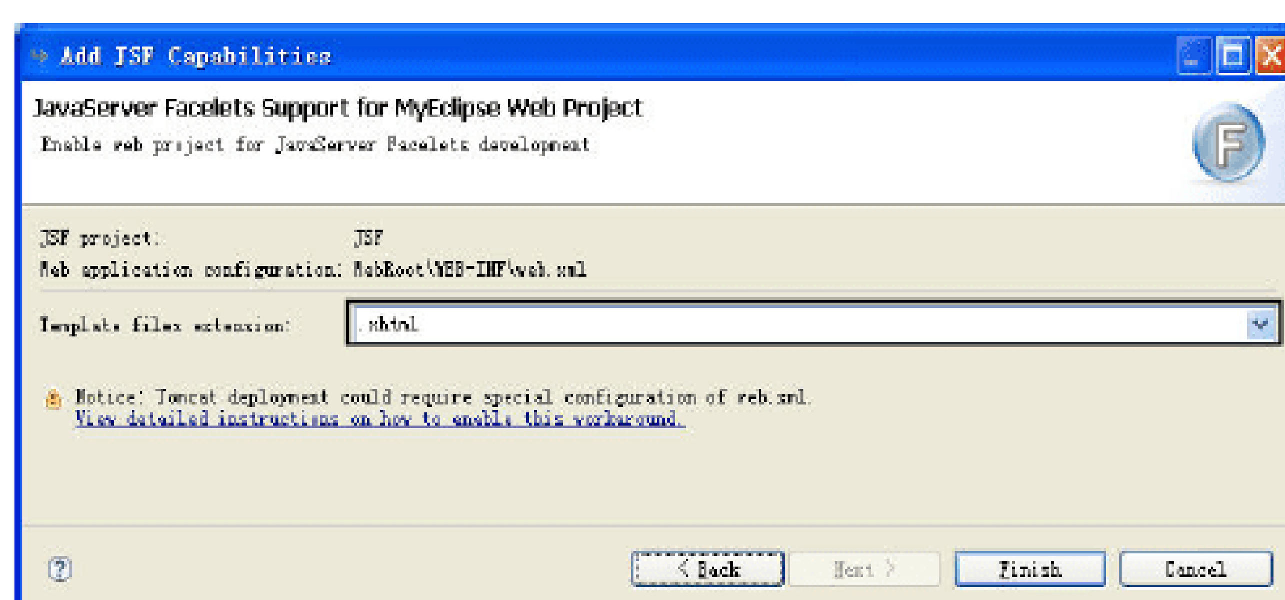


图 2.95　设置 Facelets

当完成增加 JSF 相关类库与文件操作后，展开 Package Explorer 视图中的 JSF 项目目录，如图 2.96 所示。

- ❑ Facelets 1.1 Librarise：为项目添加 JSF 类库。
- ❑ Face-config.xml：为项目添加的 JSF 配置文件。

至此，该项目已经具备了 JSF 框架的支持。

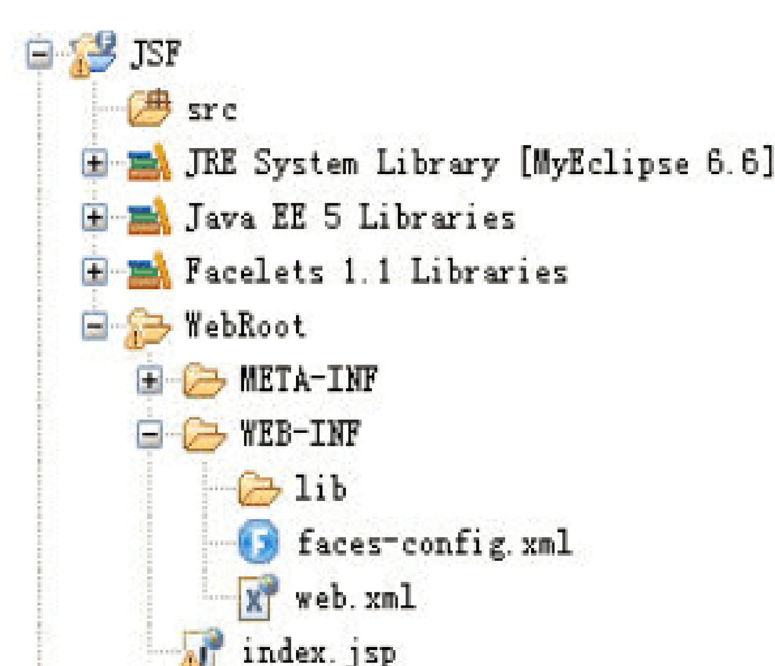


图 2.96　JSF 项目目录

2.6.2　用 MyEclipse 实现 JSF 框架项目

为了便于讲解，本节将在 2.6.1 节完成的开发环境中，利用 JSF 框架实现一个具体的应用。首先介绍代码的运行背景，实现输入数字的相加。具体步骤如下。

在介绍之前，先熟悉一下 JSF 配置文件编辑器，通过双击 faces-config.xml 文件，可以打开 faces-config.xml 配置文件编辑器，如图 2.97 所示。可以在配置文件中通过左边的工具箱创建 JSF 的各种元素，除该配置文件外也可以通过在 Outline 视图（如图 2.98 所示）中选择各种元素来创建。

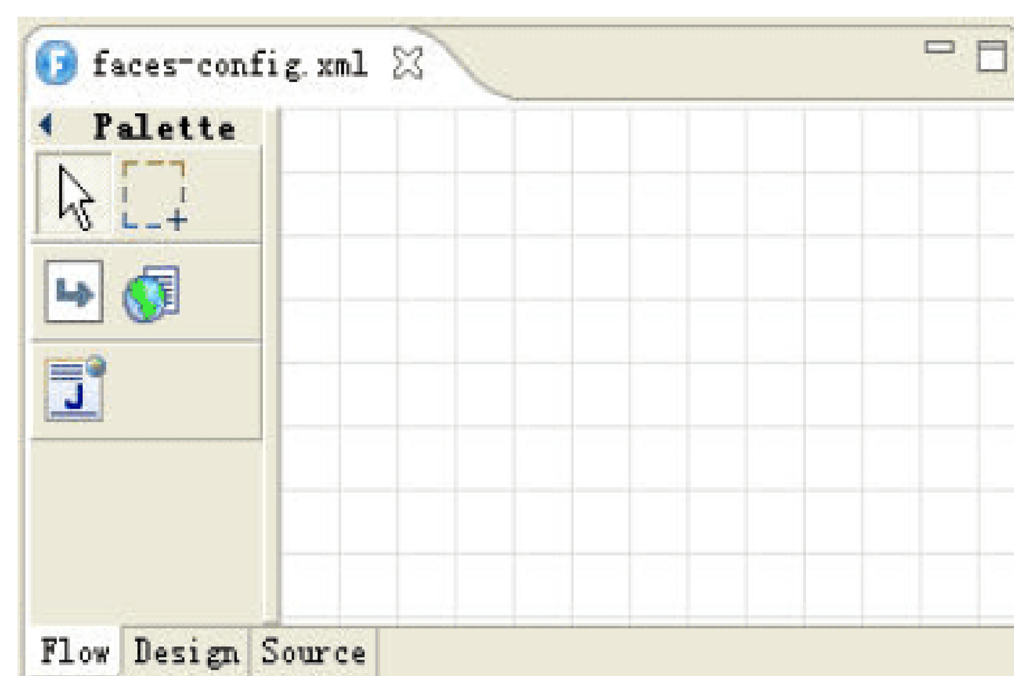


图 2.97　faces-config.xml 配置文件编辑器

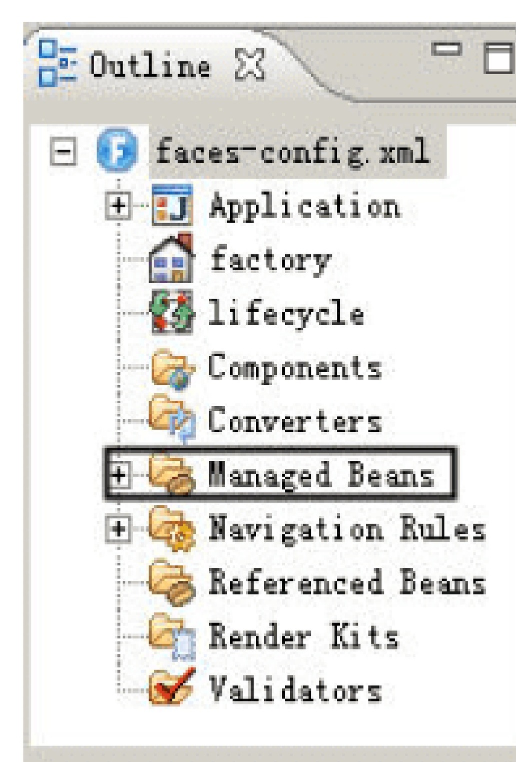


图 2.98　Outline 视图

（1）创建受管 Bean，这种 JavaBean 会从 JSP 页面表单中提取参数保存到自己属性中，然后执行参数相加的操作。首先创建名为 AddBean 的类实现了两个数相加功能，具体内容如代码 2.24 所示。

代码 2.24　两个数相加：AddBean.java

```
...
public class AddBean {
    public double add(double a,double b)
    {
        return a+b;                                          //实现两个数相加
    }
}
```

（2）接着创建一个保存页面信息和调用 AddBean 类对象名为 BackBean 的受管 Bean，具体内容如代码 2.25 所示。

代码 2.25　受管 Bean：BackBean.java

```
...
public class BackBean {
    private double firstNumber=0.0;         //定义了 firstNumber 属性
    private double secondNumber=0.0;        //定义了 secondNumber 属性
    private double result;                  //定义了 result 属性
    private AddBean addbean=new AddBean(); //创建一个 AddBean 对象
    //配置相关属性的 get()和 set()属性
...
    public String add()                     //创建一个相加方法
    {
        result=addbean.add(firstNumber, secondNumber);
                                            //利用对象 AddBean 对象实现相加
        return "success";
    }
}
```

（3）配置受管 Bean。首先在 Outline 视图中右击 Managed Beans 选项，在弹出的快捷菜单中选择 New|Managed Bean 命令（如图 2.99 所示）就会出现配置受管 Bean 对话框，如图 2.100 所示。当完成配置后，faces-config.xml 文件的内容就会多出如代码 2.26 所示的内容。

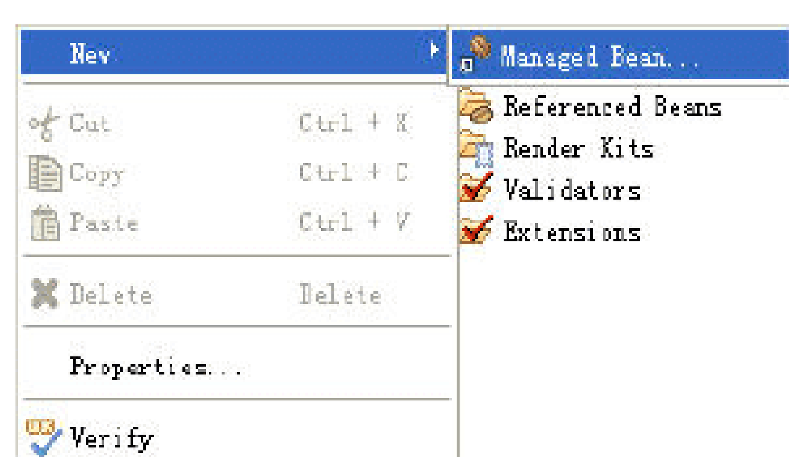


图 2.99　新建受管 Bean 菜单

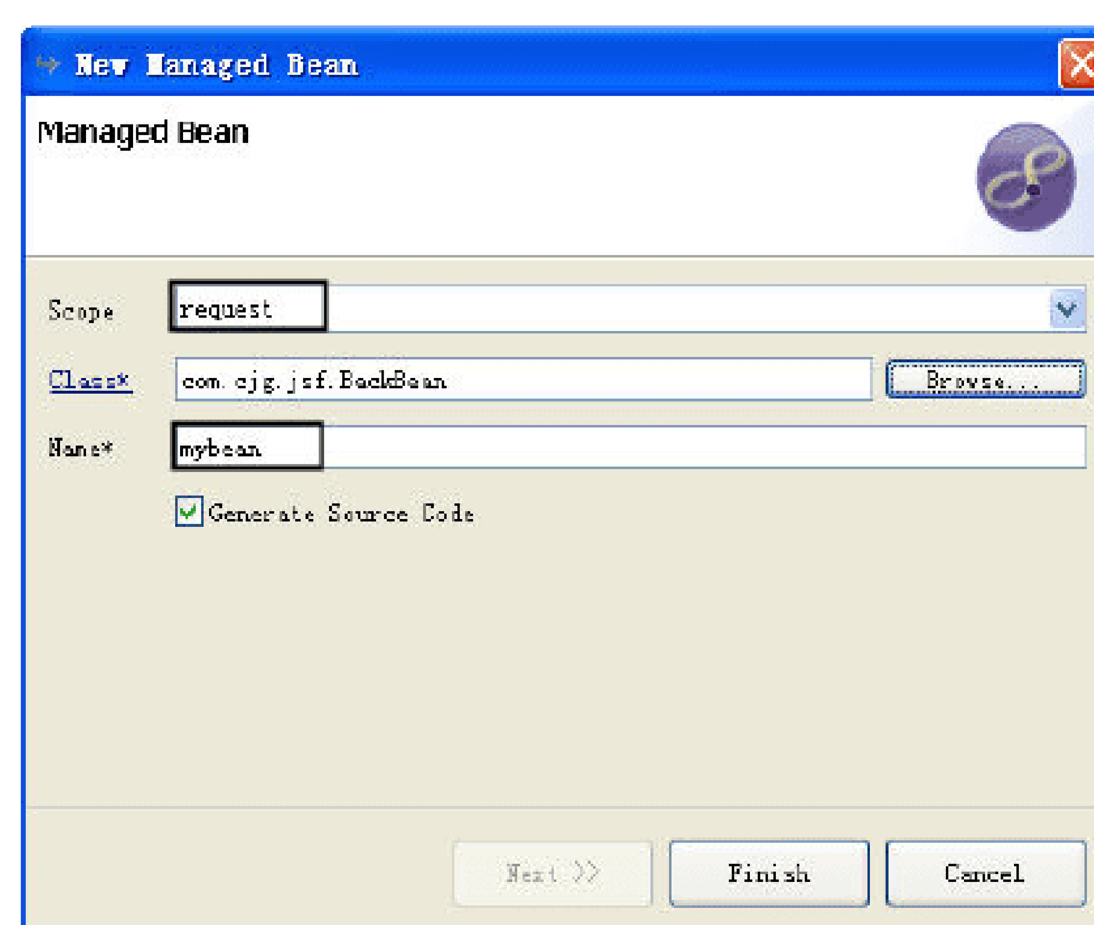


图 2.100　配置受管 Bean

代码 2.26　设置 faces-config 文件：faces-config.xml

```
<managed-bean>                              <!--配置 BackBean 类-->
    <managed-bean-name>mybean</managed-bean-name>
    <managed-bean-class>com.cjg.jsf.BackBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

【代码解析】

元素<managed-bean>表示受管理的 Bean，子元素<managed-bean-name>用来设置受管理 Bean 的引用名，子元素<managed-bean-class>用来设置受管理 Bean 的完整路径，而子元素<managed-bean-scope>用来设置受管理 Bean 的作用范围。

（4）创建 JSP 页面，在目录 JSF/WebRoot 下创建两个 JSP 页面。add.jsp 页面用来实现两个数字的输入，而 result.jsp 页面用来显示实现两个数字相加后的结果。代码 2.27 实现了该项目的输入页面。代码 2.28 为运行结果的页面。

代码 2.27　输入页面：add.jsp

```
...
<body>
    <f:view|
    <h:form|
    <h:panelGrid columns="3"|
    <h:outputLabel value="请输入第一个数字"/>          <!--显示文本-->
    <h:inputText id="firstNumber" value="#{mybean.firstNumber}"/>
                                                    <!--显示文本框-->
    <h:message for="firstNumber"/>                  <!--输入错误显示信息-->
    <h:outputLabel value="请输入第二个数字"/>          <!--显示文本-->
    <h:inputText id="secondNumber" value="#{mybean.secondNumber}"/>
                                                    <!--显示文本框-->
    <h:message for="secondNumber"/>                 <!--输入错误显示信息-->
    </h:panelGrid|
    <h:commandButton value="加" action="#{mybean.add}"/>
                                                    <!--提交调用的方法-->
    </h:form|
    </f:view|
</body>
...
```

代码 2.28　运行结果页面：result.jsp

```
<body>
    <f:view|
计算结果是:<h:outputText value="#{mybean.result}"/>  <!--显示输出文本框-->
    </f:view|
</body>
```

（5）创建导航规则。在 Outline 视图中右击 Navigation Rules 选项，在弹出的快捷菜单中选择 Rules|New 命令（如图 2.101 所示）就会出现配置导航规则对话框，如图 2.102 所示。当完成配置后，faces-config.xml 文件的内容就会多出如代码 2.29 所示的内容。



图 2.101　新建导航规则菜单



图 2.102　配置导航规则

代码 2.29　设置 faces-config 文件：faces-config.xml

```
<navigation-rule>                                    <!--配置 JSF 导航-->
    <from-view-id>/add.jsp</from-view-id>
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/result.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

上述代码表示从 add.jsp 页面出发，发出 success 信息，然后转到 result.jsp 页面。

（6）运行项目，通过在 IE 地址栏中输入 http://localhost:8080/JSF/add.faces 地址来打开输入页面，如图 2.103 所示。当在登录页面的两个文本框中输入数字“5”后，单击“加”按钮就会转到如图 2.104 所示的显示运行结果页面。

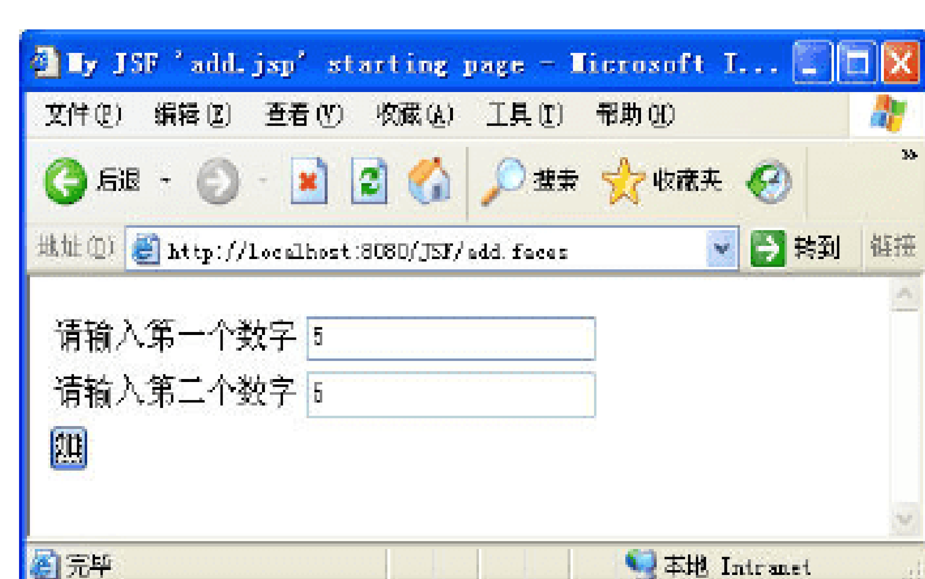


图 2.103　输入页面

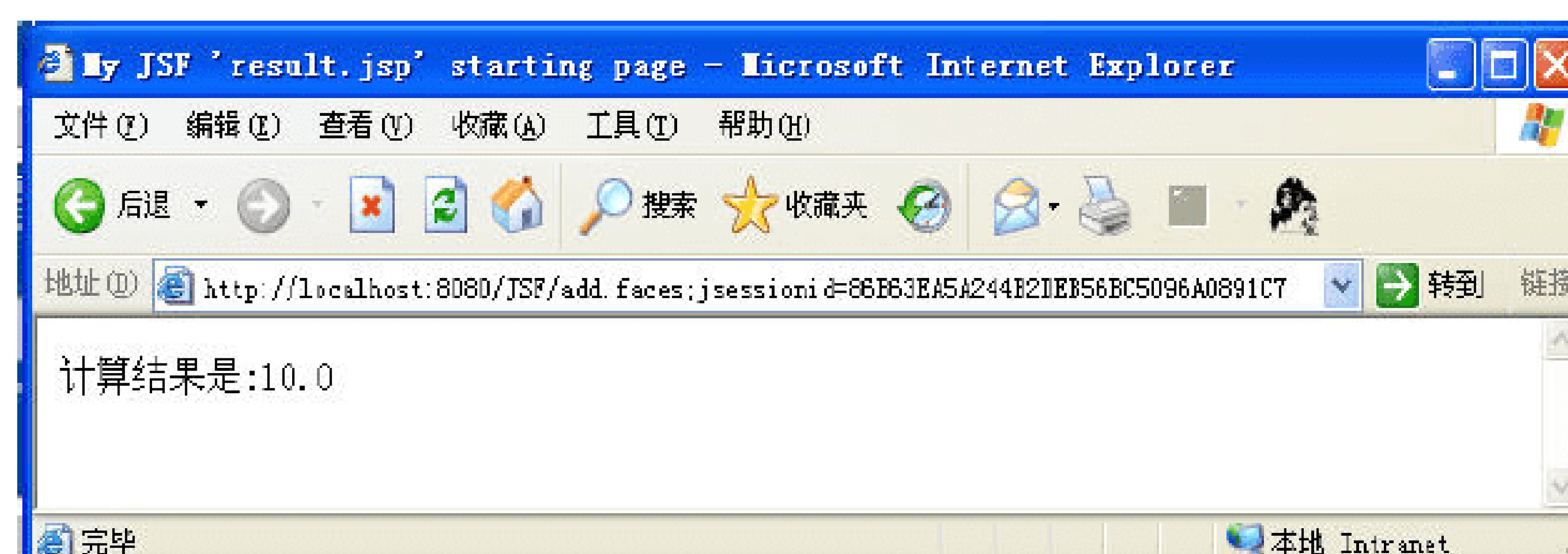


图 2.104　运行结果页面

2.7　AJAX 框架的实现

AJAX 之所以流行起来，因为其使用异步发送方式向服务器发送请求。所谓异步发送就是指用户在发送请求后，可以接着继续输入或使用该程序而不用等待服务器的响应。换句话说，就是通过 JavaScript 代码在幕后发送请求到服务器，然后服务器将数据返回给 JavaScript 代码，这样就可以迅速更新表单数据。给用户的感觉应用程序是立即完成的，即表单没有提交或刷新就得到了新数据。

2.7.1　用 MyEclipse 实现 AJAX

为了能够迅速熟悉 AJAX 技术的开发步骤和开发细节，本节将通过手工方式在开发环

境 MyEclipse 中实现一个 AJAX 应用程序，具体步骤如下。

（1）新建一个名为 AJAX 的 Web Project，详细设置如图 2.105 所示。

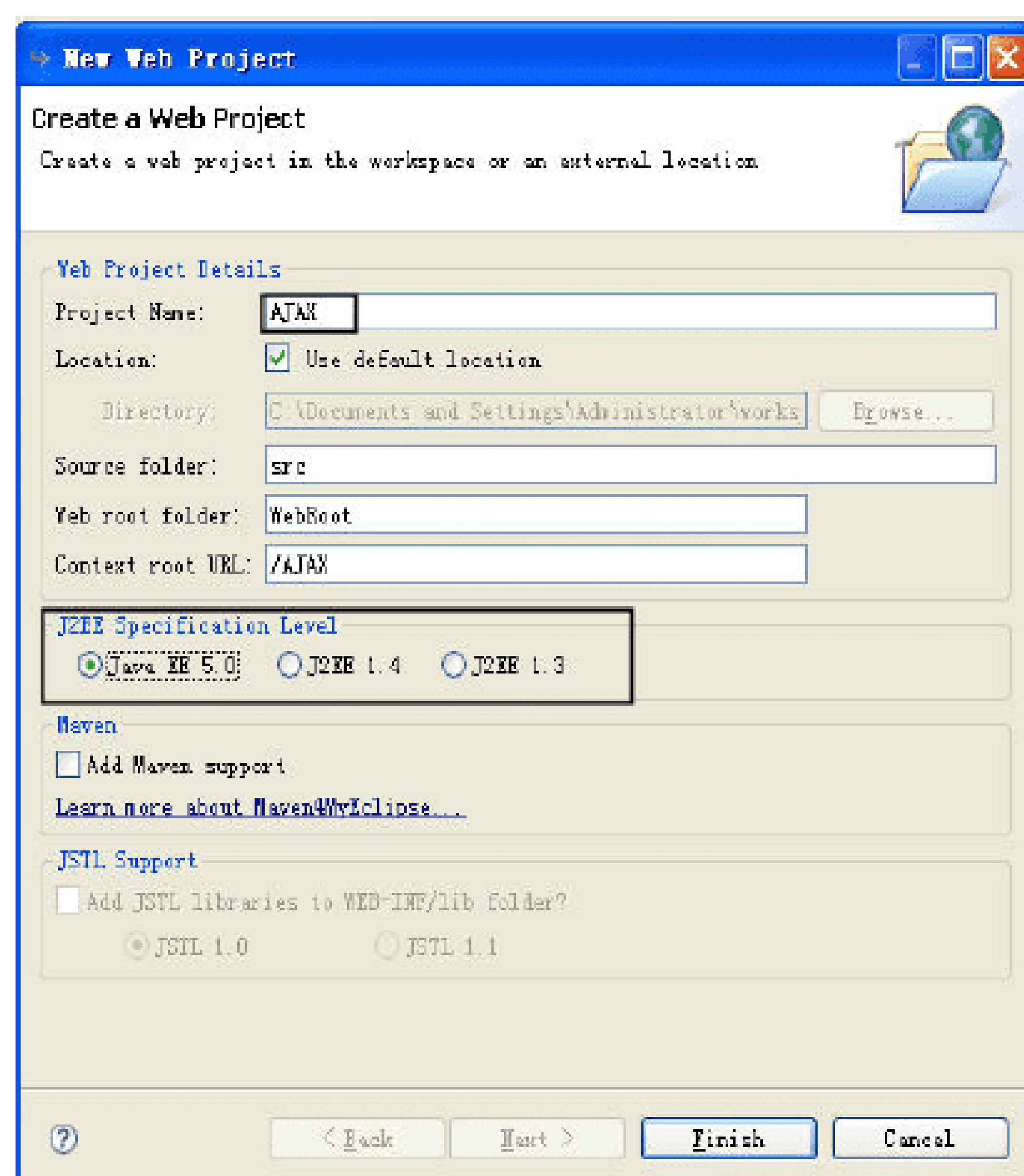


图 2.105　新建 Web Project

（2）在目录 AJAX/WebRoot 下，创建一个名为 ajax.html 的页面。代码 2.30 是信息输入页面主要部分。

代码 2.30　信息输入页面：ajax.html

```
...
    <head>
        <title>用户名校验</title>
        <script type="text/javascript" src="javascript/verifyown.js">
        </script>
    </head>
    <body>
        请输入用户名：<br />
        <input type="text" id="userName" />        <!--创建了文本框-->
        <input type="button" value="校验" onclick="verify()" />
                                                    <!--创建了按钮-->
        <div id="result"></div>
    </body>
...
```

【代码解析】

在编写 AJAX 方面的代码时，推崇使用下面的一些习惯：元素名要小写、元素标签必须关闭、属性名必须是小写的及属性值必须位于双引号中。在 AJAX 中是没有表单元素的，因为请求是不需要使用表单来进行数据提交的，而在属性方面没有 name 这个属性却有 id 属性，这是因为要使用 DOM 方式。在创建按钮时通过设置属性 onclick 的值，使得在单击按钮时执行属性 onclick 值所代表的函数。

注意：在 AJAX 中是可以出现 form 元素，只是不使用该元素的提交功能。

（3）创建一个名为 verifyjs 的 JavaScript 文件，在该文件中除了需要创建方法 onclick() 外，还要通过 XMLHTTPRequest 对象进行 AJAX 的异步数据交互。代码 2.31 实现了请求

的异步数据交互。

代码 2.31　异步数据交互：verifyjs.js

```
var xmlhttp;                                          //定义了 XMLHTTPRequest 对象
function verify() {
    var userName = document.getElementById("userName").value;
                                                      //获取文本框中的输入值
    //针对 IE 和其他类型的浏览器创建 XMLHttpRequest 对象
    if (window.XMLHttpRequest){
                                       //针对 FireFox，Mozillar，Opera，Safari，IE7，IE8
        xmlhttp = new XMLHttpRequest();
        //针对某些特定版本的 mozillar 浏览器的 BUG 进行修正
        if (xmlhttp.overrideMimeType) {
            xmlhttp.overrideMimeType("text/xml");
        }
    } else if (window.ActiveXObject) {
                                       //针对 IE6、IE5.5、IE5 浏览器来创建 XMLHttpRequest
        var activexName = ["MSXML2.XMLHTTP","Microsoft.XMLHTTP"];
        for (var i = 0; i < activexName.length; i++) {
            try{
                xmlhttp = new ActiveXObject(activexName[i]);
                break;
            } catch(e){
            }
        }
    }
    if (!xmlhttp) {             //确认 XMLHTtpRequest 对象是否创建成功
        alert("XMLHttpRequest 对象创建失败!!");
        return;
    } else {
        alert(xmlhttp.readyState);
    }
    xmlhttp.onreadystatechange = callback;                    //注册回调函数
    xmlhttp.open("GET","AJAXServer?name="+ userName,true); //设置连接信息
    xmlhttp.send(null);                                       //发送信息
}
function callback() {                                         //回调函数
    if (xmlhttp.readyState == 4) {                //判断对象的状态是交互完成
        if (xmlhttp.status == 200) {              //判断 http 的交互是否成功
            var responseText = xmlhttp.responseText;//获取服务器端返回的数据
            var divNode = document.getElementById("result");
                                                  //将数据显示在页面上
            divNode.innerHTML = responseText;     //设置元素节点中的 HTML 内容
        } else {
            alert("出错了！！！");
        }
    }
}
```

【代码解析】

- 使用 DOM 的方式获取文本框中的值时，一般会使用 getElementById()方法。该方法会返回 Id 名为传入值的元素节点，然后再通过属性获取该元素节点的属性值。
- 在创建 XMLHttpRequest 对象时，不仅要创建该对象还要检验该对象是否创建成功。创建 XMLHttpReuquest 对象是一个复杂的过程，因为浏览器会有好多类型，

而每种类型都有好多版本，针对每个浏览器的创建方式是不一样的。

- 创建完 XMLHttpRequest 对象后，就需要用到 XMLHttpRequest 对象的 onreadystatechange 属性注册回调函数。

注意：所注册的回调函数，只需要函数名而不需要括号。

（4）步骤 3 创建完后，就需要用到 XMLHttpRequest 对象的方法 open()来注册连接信息。open 方法有 3 个参数：第 1 个参数表示 HTTP 的请求方式；第 2 个参数表示请求的 URL 地址；第 3 个参数表示采用异步还是同步方式实现交互。由于 URL 的值为“AJAXServer?name=”，所以需要创建服务名为 AJAXServer 的 Servlet 程序。ajaxservlet.java 用来处理请求的信息，具体内容如代码 2.32 所示。

代码 2.32　处理请求：ajaxservlet.java

```
//引入相应包
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.URLDecoder;

public class ajaxservlet extends HttpServlet {
    protected void doPost(HttpServletRequest httpServletRequest,
            HttpServletResponse httpServletResponse) throws Servlet
            Exception,
            IOException {                                   //编写 doPost()方法
        doGet(httpServletRequest, httpServletResponse);//调用 doGet()方法
    }
    protected void doGet(HttpServletRequest httpServletRequest,
            HttpServletResponse httpServletResponse) throws Servlet
            Exception,
            IOException {                                   //编写 doGet()方法
        try {
            //设置编码格式
            httpServletResponse.setContentType("text/html;charset=utf
            -8");
            PrintWriter out = httpServletResponse.getWriter();
                                                            //获取输出流
            //获取请求参数中 name 的值
            String old = httpServletRequest.getParameter("name");
            //判断 old 的值
            if (old == null || old.length() == 0) {     //当为空时
                out.println("用户名不能为空");
            } else {                                        //当不为空
            if (name.equals("cjgong")) {
                    out.println("用户名[" + name+"]已经存在，请使用其他用户
                    名");
                } else {
                    out.println("用户名[" + name + "]尚未存在，可以使用该用户
                    名注册");
                }
            }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

接着在 web.xm 文件中配置该 Servlet 程序。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app 2 5.xsd"
    version="2.5">
    <servlet>                        <!--配置 ajaxservlet 类路径-->
        <servlet-name>ajaxservlet</servlet-name>
        <servlet-class>ajaxservlet</servlet-class>
    </servlet>
    <servlet-mapping>                 <!--配置 ajaxservlet 映射路径-->
        <servlet-name>ajaxservlet</servlet-name>
        <url-pattern>/ajaxservlet</url-pattern>
    </servlet-mapping>
</web-app>
```

（5）创建好连接服务器的信息后，就需要用到 XMLHttpRequest 对象的 send()方法来发送数据，与服务器端进行交互。

（6）最后，如何编写回调函数 callback()呢？首先根据 XMLHttpRequest 对象的属性 readyState 来判断对象状态是否交互完成，如果完成接着再根据 XMLHttpRequest 对象的属性 status 来判断 HTTP 的交互是否成功。当上面的这些都成功时，就把获取服务器端的纯文本数据显示在相应的 Id 元素所代表的元素上。

（7）单击工具栏上的按钮，把该项目发布到服务器上。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8081/AJAX/ajax.html，就会打开首页，在该页面输入 cjgong 字符串（如图 2.106 所示）。单击“校验”按钮则会出现如图 2.107 所示的页面。否则就会出现如图 2.108 所示的页面。



图 2.106　首页

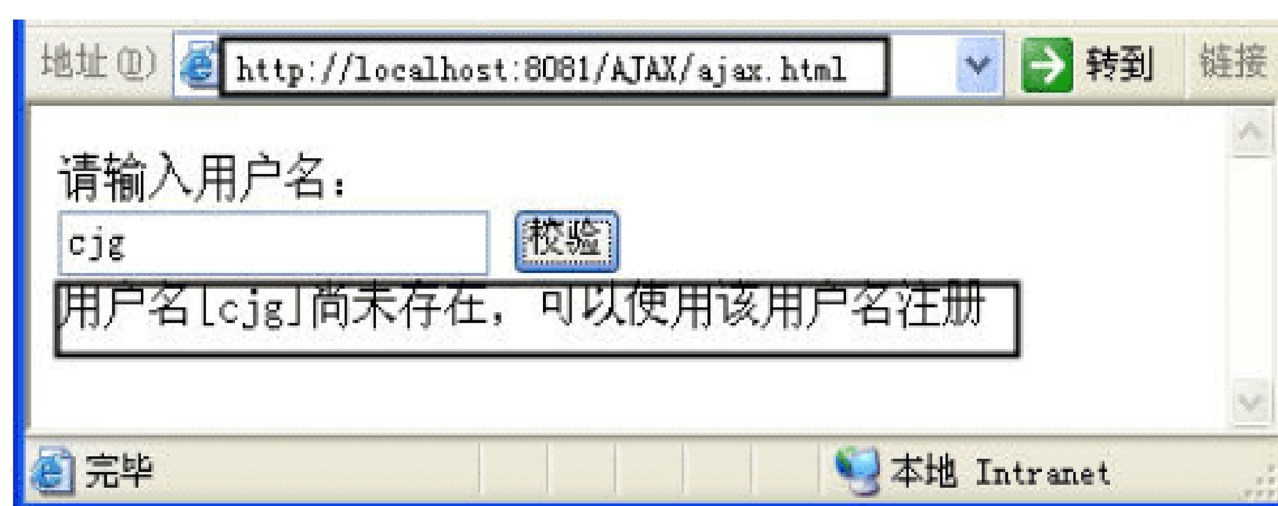


图 2.107　注册过用户名

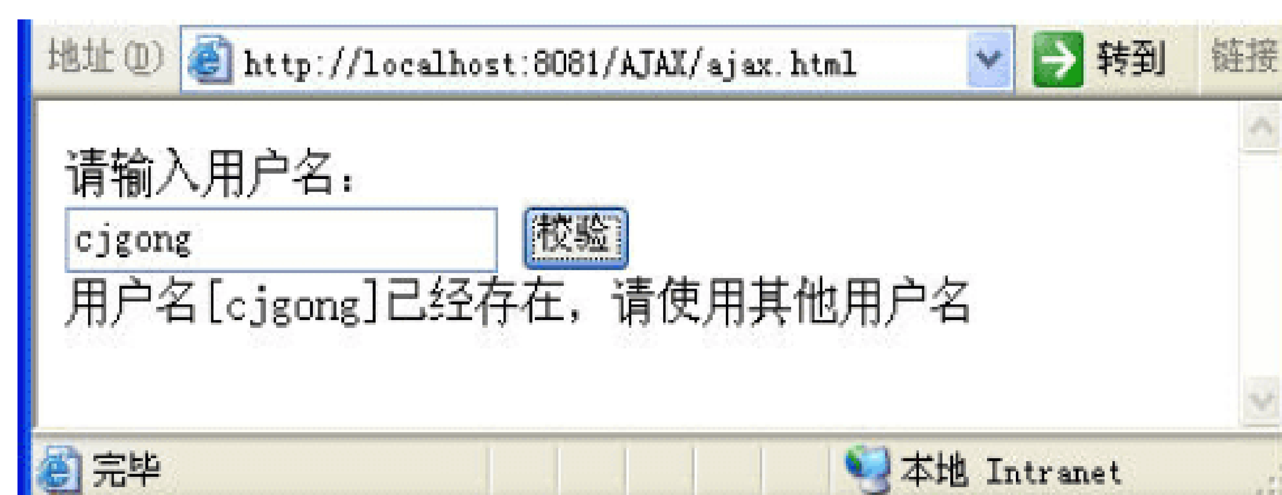


图 2.108　没注册用户名

注意： 比较上述 3 个运行结果的地址，可以发现没有发生变化。这就是所谓的 AJAX 技术。

2.7.2　分析 AJAX 技术

通过 2.7.1 节开发步骤基本了解 AJAX 技术的运行过程，AJAX 技术中的 XMLHttpRequest 对象发送请求给服务器，服务器处理后以 XML 数据或文本方式把结果返回给 AJAX，最后再把结果以 HTML+CSS 的形式显示在浏览器上，整个过程如图 2.109 所示。

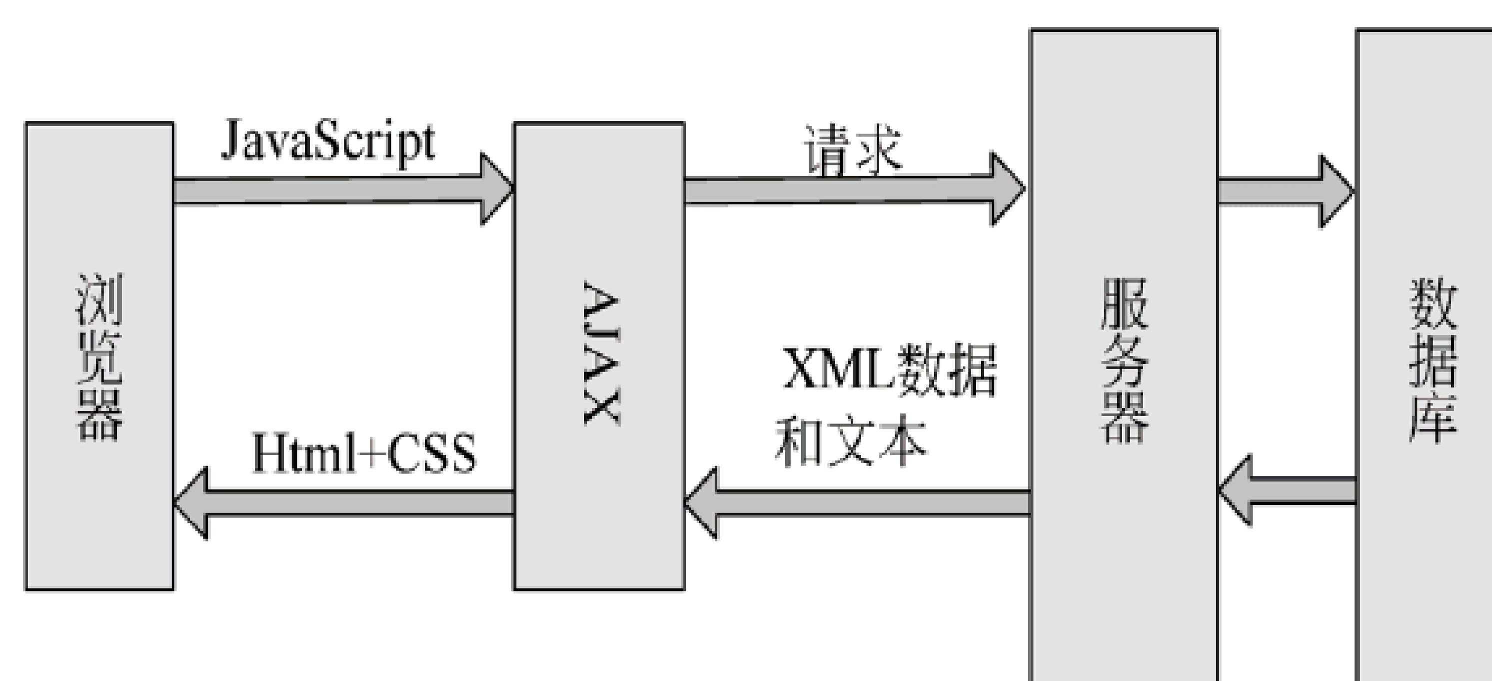


图 2.109　AJAX 运行流程

在开发 AJAX 方面的程序时，创建 XMLHttpRequest 对象是一个非常复杂的过程，因为浏览器的种类和版本太多，但是创建完 XMLHttpRequest 对象后，剩下的过程就非常机械化。这些雷同的流程如下。

- 利用 DMO 方式从 Web 表单中获取需要的数据。
- 为 open()方法建立要连接的 URL。
- 利用 open()方法打开到服务器的连接。
- 设置服务器在完成后要运行的函数，即回调函数。
- 利用 sent()方法发送请求。

综上所述，在 AJAX 技术中，除了 XMLHttpRequest 对象感到陌生外，而其他的技术都是非常熟悉的技术。例如用来建立 Web 表单的 HTML 语言、帮助改进与服务器应用程序通信的 JavaScript 语言、用于动态更新表单的 DHTML 或 Dynamic HTML 技术和用来处理 HTML 结构的文档对象模型 DOM。

2.8　使用 JDBC 连接数据库

JDBC（Java Database Connectivity）全称叫数据库连接，其是用来执行 SQL 的 Java API。开发人员可以使用这些标准 API，来连接和操作数据库，实现数据库应用程序的开发。

2.8.1　JDBC 的基本概念

当开发人员使用 JDBC 连接数据库时，只需要调用 JDBC API，就可以向任何数据库

发送 SQL 语句执行相应功能，而详细的连接过程就没有必要手动编写了。JDBC 的工作原理如图 2.110 所示。

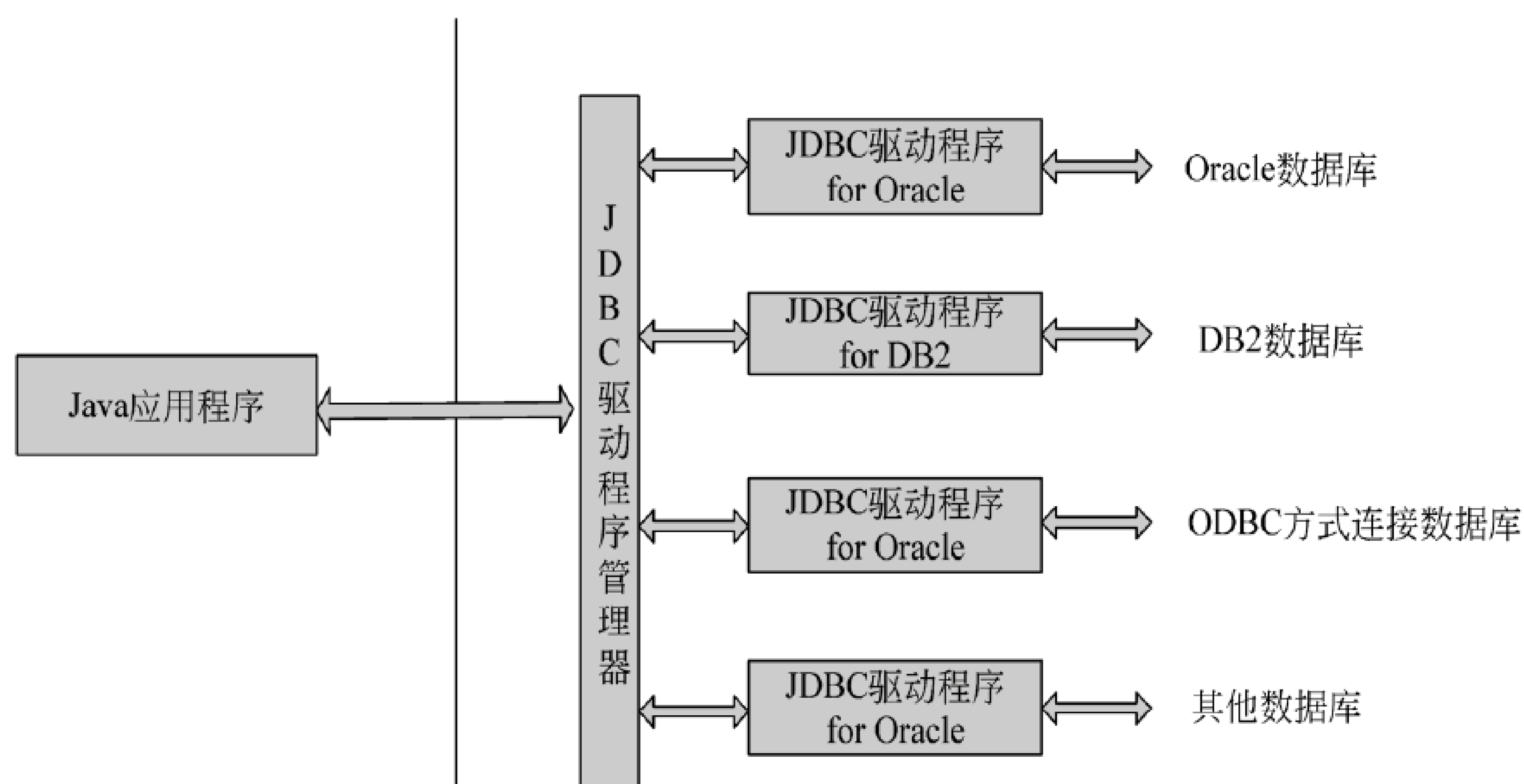


图 2.110　JDBC 的工作原理

Java 应用程序不会直接操作底层的数据库，而是通过 JDBC API 的 JDBC 驱动程序管理器，委托给底层各种类型的 JDBC 驱动程序来操作数据库。JDBC 驱动程序可以分为 4 类。

- JDBC-OCBC 桥：Java 应用程序数据操作指令在 JDBC 驱动程序管理器的管理下，经过 JDBC-OCBC 桥驱动转换 ODBC 驱动程序的指令格式，接下来再通过 ODBC 的方式连接数据库。
- Java 到本地 API：Java 应用程序数据操作指令在 JDBC 驱动程序管理器的管理下，转换为调用本地 API。这里所说的本地 API 是指计算机中安装数据库客户端的 API。
- Java 到网络协议：通过网络协议（与具体数据库无关）把 Java 应用程序数据操作指令发送给数据库服务器。
- Java 到数据库协议：通过特定数据库网络协议，把 Java 应用程序数据操作指令发送给特定类型数据库服务器。

在 JDBC 驱动程序中的第一种方式中出现了 ODBC 驱动程序，什么是 ODBC 驱动程序？ODBC 全称开放式数据库互连（Open Database Connectivity），该驱动程序是由微软设计和开发的一种通用、标准操作数据库的 API，可以说其是一种数据库系统应用程序的接口规范。该种驱动程序的工作原理如图 2.111 所示。

ODBC 驱动程序的作用，是把应用程序中操作数据库的指令，转换成与某一种数据库相关的数据库指令，然后由具体相关的驱动程序传送给该数据库。在开发的时候，只需要面对统一格式的 ODBC 数据源就可以，而不必针对各种不同数据库做各种设计。

综上所述，JDBC 可以实现如下功能：

- 提供多样化的数据库连接方法。
- 为各种不同的数据库提供统一的操作界面。

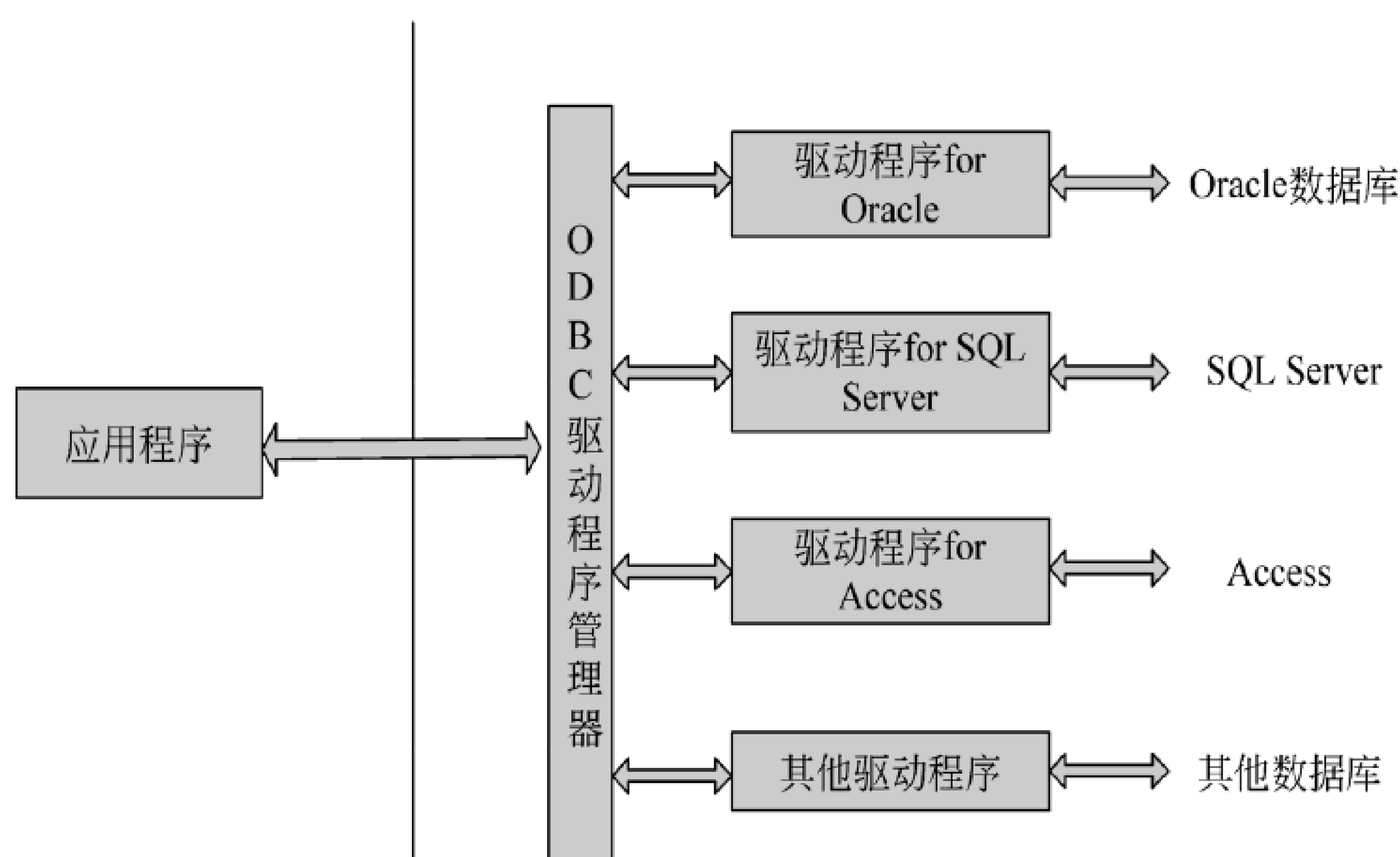


图 2.111　ODBC 的工作原理

2.8.2　JDBC 的基本步骤

使用 JDBC 连接数据库存取数据时，必须要执行 3 个步骤。

（1）加载及注册适当的 JDBC 驱动程序。

（2）建立到指定数据库的连接对象。

（3）提交数据库查询和取得查询对象。

JDBC 与数据库建立连接的第一个步骤为加载适当的驱动程序，一般使用如下代码：

```
Class.forName()
```

当加载完驱动程序后，驱动程序都会创建一个 Driver 对象，并且会让 DriverManager.registerDriver()自动注册此对象。

在 JDBC 中使用数据库 URL 来标识目标数据库，其基本语法格式如下：

```
Jdbc:<子协议名|:<子名称|
```

- jdbc 为协议名，一般保持不变。
- <子协议名|指定目标数据库的种类和具体连接方式。
- <子名称|指定具体的数据库/数据源连接信息，例如数据库服务器的 IP 地址/通信端口号、ODBC 数据源名称、连接用户名/密码等。

注意：一般子名称的格式和内容会随着子协议名的不同而不同。

下面是一些连接数据库的常见语法格式：

（1）假如使用 MySQL 的驱动程序来连接 MySQL，则语法格式如下：

```
Jdbc:mysql://hostname:port/databasename?param=valuel
```

其中子协议名为：mysql，子名称：//hostname:port/databasename?param=valuel。

在子名称中，hostname：表示主机名称，其也可以为一个有效 IP 地址；Port：表示连接数据库使用的端口；databasename：表示连接数据库的名字；param：表示参数的设置，

可以有好多个。

```
Jdbc:mysql://localhost:3306/sample?user=root&password=root
```

上述代码表示连接到一个本地且端口号为 3360 的 MySQL 服务器，并且指定连接的数据名为 sample，“?”后的参数代表登录此数据库的名称和密码。

（2）假如使用 JDBC-ODBC bridge driver 驱动程序来连接名为 sample 的本地数据库，则语法格式如下：

```
Jdbc:odbc:sample?user=root&password=root
```

其中子协议名为：odbc，子名称：sample?user=root&password=root。

（3）假如使用 Oracle 驱动程序来连接名为 sample 的本地数据库，则语法格式如下：

```
Jdbc:oracle:thin://127.0.0.1:1521: sample
```

其中子协议名为：oracle:thin，子名称：//127.0.0.1:1521: sample。

（4）假如使用 SQL Server 驱动程序来连接名为 sample 的本地数据库，则语法格式如下：

```
Jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=sample
```

其中子协议名为：microsoft:sqlserver，子名称：//127.0.0.1:1433;databasename=sample。

当加载完驱动程序并且用数据库 URL 标示一个数据源后，现在就可以建立一个连接对象，如下所示：

```
String url=" Jdbc:mysql://localhost:3306/sample?user=root&password=root";
Connection con = DriverManager.getConnection(url);
```

上述代码也可以改写成如下形式：

```
String url="Jdbc:mysql://localhost:3306/sample";
String user="root";
String password="root";
Connection con = DriverManager.getConnection(url,user,password);
```

从上可以看出，连接对象主要是通过 DriverManager.getConnection()来实现。当顺利取得连接对象后，就可以由它来创建一个陈述对象，陈述对象主要用来传送 SQL 语句到数据库服务器，然后由该数据库执行 SQL 语句。例如下面产生一个名叫 stmt 的陈述对象。

```
Statement stmt= con.createStatement();
```

在 Statement 类中，一般只会涉及 3 种执行语句方法，分别如下。

- ❑ executeQuery()：执行 SQL 查询使用的方法。
- ❑ executeUpdate()：执行 SQL 更新使用的方法。
- ❑ execute()：当不知道正要执行的 SQL 是查询还是更新情况下使用的方法。

当使用 executeQuery()方法时会返回一个 ResultSet 对象。当向数据库查询数据的时候，数据库会回传一个结果集合到 JDBC 中，这个结果集合就是所谓的 ResultSet 对象。

本节介绍的类分别来自 JDBC API 中，它们的运行原理如图 2.112 所示。

- ❑ Java.sql.DriverManager：被用来作为初始化的一部分，提供管理 JDBC 驱动器设置的基本服务。

- ❑ Java.sql.Connection：用来表示与一个特定数据库的会话。
- ❑ Java.sql.Statement：用来执行 SQL 语句并获取它产生的结果。

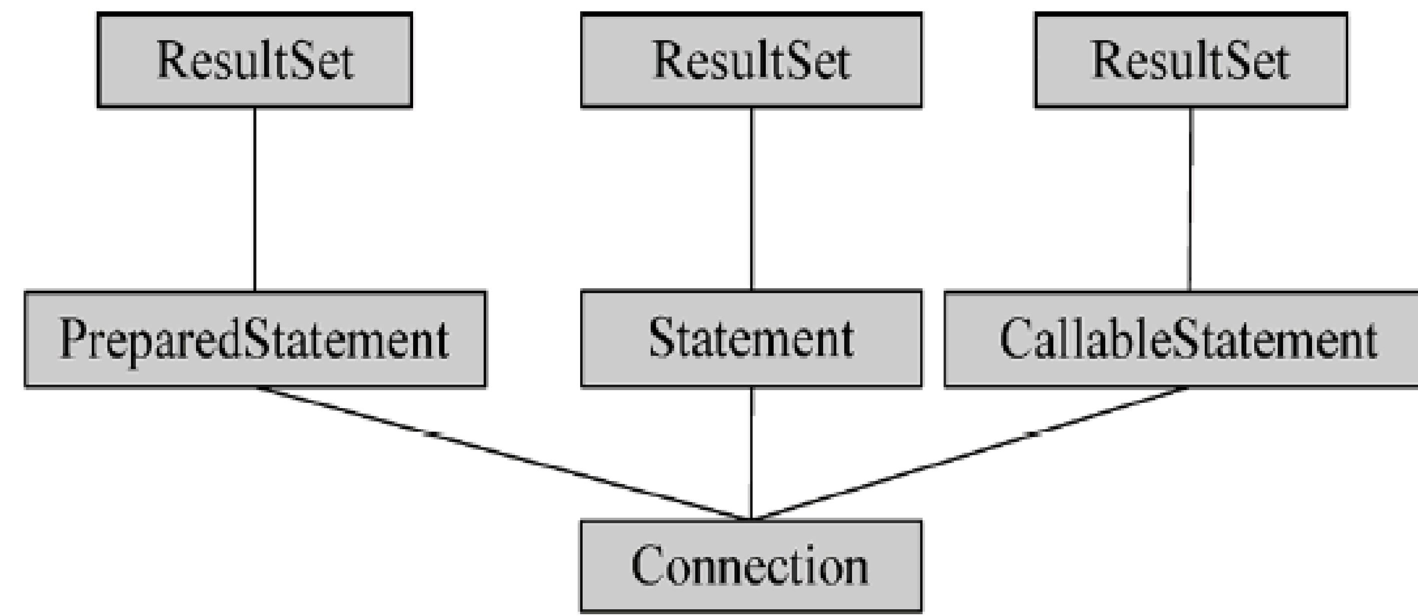


图 2.112　运行原理

2.9　小　　结

本章主要介绍了 MyEclipse 开发工具对框架的支持，其中包括 Struts 1.x 和 Struts 2.x、Hibernate、JPA、Spring 和 JSF 框架。对于这些框架，MyEclipse 开发工具不仅实现全方位支持，而且还提供了图形向导。于是使用 MyEclipse 开发工具开发 Java Web 项目是一个非常不错的选择。

当 Java Web 项目不需要使用具体框架时，经常会使用 JSP+Servlet+JavaBean 框架来作为开发项目的解决方案，所以 JSP+Servlet+JavaBean 框架是开发 Java Web 项目和使用其他框架的基础。

在 Java Web 项目中免不了需要使用 AJAX 技术和连接数据库，所以 AJAX 技术和连接数据库的 JDBC 技术同样也是非常重要。

第 3 章　实现各种框架的集成

在开发具体项目时，一般会使用关于 SSH 框架集成的解决方案。所谓 SSH 集成，就是指 Struts、Spring 和 Hibernate 框架的集成。本章将详细讲解 Struts、Spring 和 Hibernate 框架的集成原理和详细过程。本章不仅介绍了 SSH 框架集成（其中 Struts 框架不仅可以是 Struts 1.x 框架，而且还可以是 Struts 2.x 框架），而且还实现了与其他框架的集成。这些集成框架分别为：

- Spring+Struts 1.x 和 Spring+Hibernate；
- Struts 1.x+Spring+Hibernate；
- Spring+Struts 2.x；
- Struts 2.x+Hibernate 和 Struts 2.x+JPA；
- Struts 2.x+Spring+Hibernate。

3.1　Spring 框架与其他框架的集成原理

Spring 框架最有价值的一个提议就是允许选择，即其不会强迫使用特定的架构。在 Java Web 领域，Spring 框架不仅提供了自己的 Web 框架（Spring MVC），而且还提供了与其他流行的 Web 框架集成的能力。

本节将介绍如何用 Spring 框架集成各种流行的框架，分别为：

- Spring 框架与 Struts 1.x 框架的集成；
- Spring 框架与 Hibernate 框架的集成。

3.1.1　依赖查找方式实现 Spring 与 Struts 集成

如何实现 Spring 框架与 Struts 1.x 框架的集成呢？在 Spring 中最重要的是获取到 BeanFactor，只有得到该对象才能从 IOC 容器中获取业务对象，调用业务方法。所以应该在 Struts 1.x 框架的 Action 上实现两者的集成。

为什么叫依赖查找方式，因为在 Struts 1.x 的 Action 中，是通过依赖查找方式取得 BeanFactory。只要获取 BeanFactory 对象，就可以从 IOC 容器中获取业务对象，然后调用业务方法。下面将通过一个具体的事例来介绍如何实现 Spring 框架与 Struts 1.x 框架的集成，具体步骤如下。

首先介绍代码的运行背景，把登录页面输入的信息经过简单处理后显示出来。在 MyEclipse 开发环境中无论先实现 Struts 1.x 框架还是 Spring 框架都一样，各个框架实现的详细步骤可以查看第 2 章，本节将不详细介绍。

（1）首先建立两个页面：首页 index.jsp 和登录页面 login.jsp，这两个页面的主要内容分别如代码 3.1 和代码 3.2 所示。

代码 3.1　首页：index.jsp

```
...
<body><a href="logininput.do">登录</a>                          //链接
</body>
...
```

代码 3.2　登录页面：login.jsp

```
...
<body>
    <form action="login.do" method="post">
        用户：<input type="text" name="username"><br>              //文本框
        密码：<input type="password" name="password"><br>          //密码框
        <input type="submit" value="登录">
    </form>
</body>
...
```

（2）新建 FromBean，当登录页面 login.jsp 发出请求后，其请求的信息会被封装到 From 中。在包 com.cjg.user.forms 中建立 LoginActionForm.java 文件，代码 3.3 实现对信息的封装。

代码 3.3　封装信息：LoginActionForm.java

```
...
public class LoginActionForm extends ActionForm {
    private String username;                          //创建了用户名属性
    private String password;                          //创建了密码属性
    //省略上述属性的 set() 和 get() 方法
    ...
}
```

（3）编写业务层，对 LoginActionForm 中封装的信息实现相应的功能，在包 com.cjg.user.manager 中建立接口 UserManager，该接口只定义实现登录功能的方法，然后创建出一个名为 UserManagerAdd.java 的管理类来继承该接口类。代码 3.4 实现用户名字的输出。

代码 3.4　用户管理：UserManagerAdd.java

```
...
public class UserManagerAdd implements UserManager {
    public void login(String username, String password) { //实现登录方法
        System.out.println("UserManagerImpl.login() -- username=" +
        username);                                          //输出用户名
    }
}
```

对于业务层除了编写相应的功能外，还必须把业务层对象配置到 Spring 配置文件 applicationContext-beans.xml 中，其代码如代码 3.5 所示。

代码 3.5　Spring 配置文件：applicationContext-beans.xml

```
...
<beans>                                   <!--配置 userManager 类-->
    <bean id="userManager" class="com.cjg.user.manager.UserManagerAdd"/>
</beans>
...
```

（4）新建 Action，在包 com.cjg.user.actions 创建类 LoginAction.java。代码 3.6 实现对请求信息操作。

代码 3.6　操作用户请求：LoginAction.java

```
...
public class LoginAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
        LoginActionForm laf = (LoginActionForm) form;  //获取 ActionForm
        BeanFactory factory = WebApplicationContextUtils
                .getRequiredWebApplicationContext(request.getSession()
                        .getServletContext());          //创建 BeanFactory
        //获取 UserManager 对象
        UserManager userManager = (UserManager) factory.getBean("user
        Manager");
        userManager.login(laf.getUsername(), laf.getPassword());
                                                       //输出用户名
        return mapping.findForward("success");         //转向成功页面
    }
}
...
```

【代码解析】

- 在编写上述代码时，绝不能使用如下方式来创建对象 BeanFactory。

```
BeanFactory factory = new ClassPathXmlApplicationContext("application
Context-beans.xml");
```

这是因为每次请求时都会创建一个 BeanFactory 对象，显然这是不符合要求的。为了解决 BeanFactory 问题，可以使用 Spring 框架的 listener 类。该类首先会读取 Sping 的配置文件，然后会根据配置文件创建一个对象 BeanFactory（注意这个对象需要放在 ServletContext 中）。

- listener 类只是把对象存储到 ServletContext 对象中，如何读取出来呢？Spring 提供了一个辅助类 WebApplicationContextUtils，通过该类的方法 getRequiredWeb-ApplicationContext()就可以实现该功能。由于在 getRequiredWebApplicationContext()方法中，需要一个 ServletContext 类型参数，而 request.getSession().getServletContext()就可以得到当前 ServletContext。这就是文件 LoginAction.java 中出现下面代码的原因。

（5）LoginAction 类成功执行的前提条件，就是要在配置文件 web.xml 中对该类实现配置，其内容如代码 3.7 所示。

代码 3.7　web.xml 配置文件：web.xml

```
...
    <context-param>              <!--配置 contextConfigLocation 的相关参数-->
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:applicationContext-*.xml</param-value>
    </context-param>
    <listener>                   <!--配置侦听器-->
    <listener-class>org.springframework.web.context.ContextLoader
    Listener</listener-class>
    </listener>
...
```

【代码解析】

在上述代码中，元素<context-param>用来设置配置文件的位置，而元素<listener>的作用则是通过指定的 Sping 框架中的类来实现配置文件的读取。

注意：元素<context-param>中子元素<param-name>的值不能改变。

（6）当 LoginAction 处理完后，就会转向相应的页面，即页面 success.jsp。代码 3.8 是成功页面的主体部分。

代码 3.8　成功页面：success.jsp

```
...
<body>
    ${loginForm.username },用户登录成功!
</body>
...
```

（7）在该项目中涉及到 3 个配置文件：struts-config.xml、applicationContext-beans.xml 和 web.xml。其中 struts-config.xml 配置文件主要用来配置关于 Struts 的信息，具体内容如代码 3.9 所示。

代码 3.9　struts 配置文件：struts-config.xml

```
...
<struts-config>
    <form-beans>                     <!--配置 ActionForm-->
        <form-bean name="loginForm" type="com.cjg.user.forms.LoginAction
        Form"/>
    </form-beans>
    <action-mappings>                <!--配置程序流程-->
        <action path="/logininput"
                forward="/login.jsp">
        </action>
        <!--配置关于 login 的 action-->
        <action path="/login"
                type="com.cjg.user.actions.LoginAction"
                name="loginForm"
                scope="request">
            <forward name="success" path="/success.jsp"/>
        </action>
    </action-mappings>
   <message-resources parameter="MessageResources" />
```

```
</struts-config>
...
```

【代码解析】

上述代码首先配置了 ActionFrom 即代码中的 loginForm。接着配置 Action，在元素<action>中，属性 path 的值表示如何来访问这个 Action，注意值是以“/”开始。如果在 Action 中使用了 ActionFrom，就要设置属性 name 的值为 loginForm。最后在 action 元素的子元素<forward>中，设置当返回值为 success 时，则转到页面 success.jsp 上。

编写完该项目后，单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/springstruts/index.jsp，运行结果如图 3.1 所示。单击“登录”链接就进入如图 3.2 所示的登录页面，在该页面的用户和密码文本框中填写相应的信息后，单击“登录”按钮后就进入如图 3.3 所示的登录成功页面。



图 3.1　index.jsp 页面

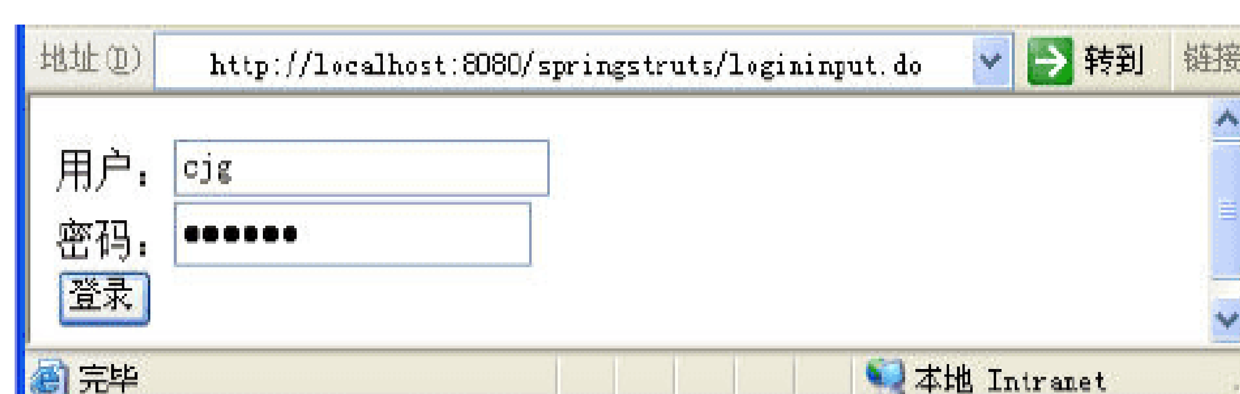


图 3.2　登录页面

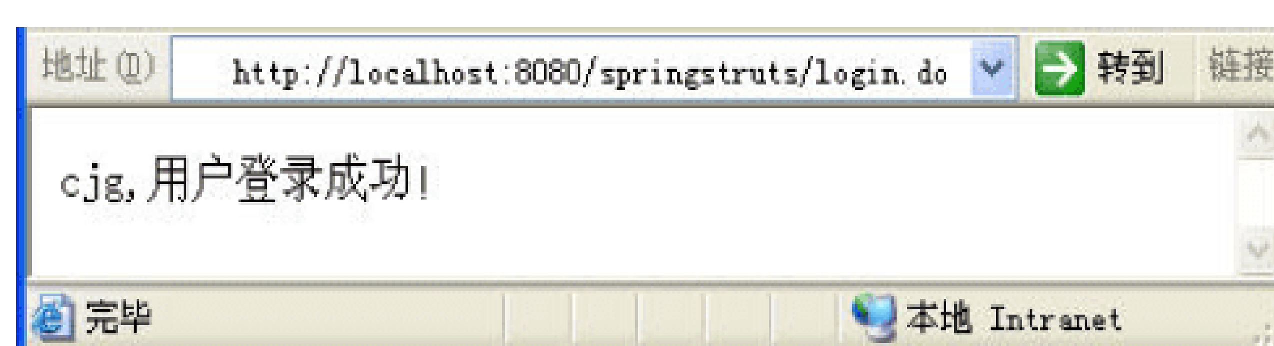


图 3.3　成功页面

利用依赖查找方式实现 Spring 和 Struts 1.x 的集成的解决方案中，由于在 Action 类中通过 BeanFactory 来实现各种获取，所以会存在依赖关系。

3.1.2　Action 注入方式实现 Spring 与 Struts 集成

依赖查找方式实现 Spring 与 Struts 1.x 的集成方案虽然容易使用，但是该方案有依赖性。即在 Action 中直接使用了依赖查找，使得 Action 离不开 BeanFactory 对象。为了避免该弊端，在以 Action 注入方式实现 Spring 与 Struts 1.x 集成的方案中，业务逻辑对象通过 Spring 注入到 Action 中，从而避免了依赖性。

本节将通过 Action 注入方式实现 Spring 与 Struts 1.x 集成方案来实现上一节的功能，接着 3.1.1 节的第（3）步骤后开始编写第（4）步骤，具体为：

（4）在包 com.cjg.user.actions 中建立类 LoginAction.java 实现对请求信息操作，代码 3.10 实现对请求信息操作。

代码 3.10　操作用户请求：LoginAction.java

```
...
public class LoginAction extends Action {
    private UserManager userManager;                //创建用户管理属性
```

```
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
        LoginActionForm laf = (LoginActionForm)form;
                                        //获取 ActionAction
        userManager.login(laf.getUsername(), laf.getPassword());
                                        //实现用户管理
        return mapping.findForward("success"); //转向成功页面
    }
    public void setUserManager(UserManager userManager) {
                                        //设置用户管理属性
        this.userManager = userManager;
    }
}
...
```

【代码解析】

为了把对象 userManager 通过 Spring 注入到类 LoginAction 中的，必须要创建一个 userManager 属性。为了解决 Action 中依赖关系，可以把该 Action 注入到 Spring 的 Ioc 中，这样 Ioc 容器就可以创建和管理 Action，从而消除 Action 中的依赖关系。代码 3.11 为 applicationContext-beans.xml 配置文件代码。

代码 3.11　Spring 配置文件：applicationContext-beans.xml

```
...
<beans>                      <!--配置类 LoginAction -->
    <bean name="/login" class="com.cjg.user.actions.LoginAction" scope=
    "prototype">
        <property name="userManager" ref="userManager"/>
    </bean>
</beans>
...
```

注意：由于其他内容跟 3.1.1 节中的第（4）步一样，所以这里省略。

（5）登录页面发出请求，由于首先会到 struts-config.xml 配置文件中找 path 属性为/login 的元素<action>，然后查看该元素的 type 属性的值是否实例化。如果没有实例化，Struts 1.x 框架会自动实例化。所以应该使用 Spring 中关于 Action 的代理类 ActionProxy 来实现 Action 的注入。当使用 ActionProx 时，必须对 struts-config.xml 文件实现如代码 3.12 所示的配置。

代码 3.12　struts 配置文件：struts-config.xml

```
...
<struts-config>
    <form-beans>                        <!--配置 formbean-->
        <form-bean name="loginForm" type="com.cjg.user.forms.Login
        ActionForm"/>
    </form-beans>
    <action-mappings>                   <!--配置 action-->
        <action path="/logininput"      <!--配置关于 logininput 的 action-->
                forward="/login.jsp"
        ></action>
        <!--配置关于 login 的 action-->
        <action path="/login"
```

```
                type="org.springframework.web.struts.Delegating
                ActionProxy"
                name="loginForm"
                scope="request">
            <forward name="success" path="/success.jsp"/>
        </action>
    </action-mappings>
    <message-resources parameter="MessageResources" />
</struts-config>
...
```

【代码解析】

在上述代码中登录页面发出请求后，首先会到 struts-config.xml 配置文件中找到 path 属性为/login 的元素<action>，然后把请求交给代理类 org.springframework.web.struts.DelegatingActionProxy 来处理。接着代理类通过自己创建的 BeanFactory 对象，根据 Spring 配置文件中关于“name=/login”的信息，创建出真正的 Action 文件。该解决方案的流程图如图 3.4 所示。

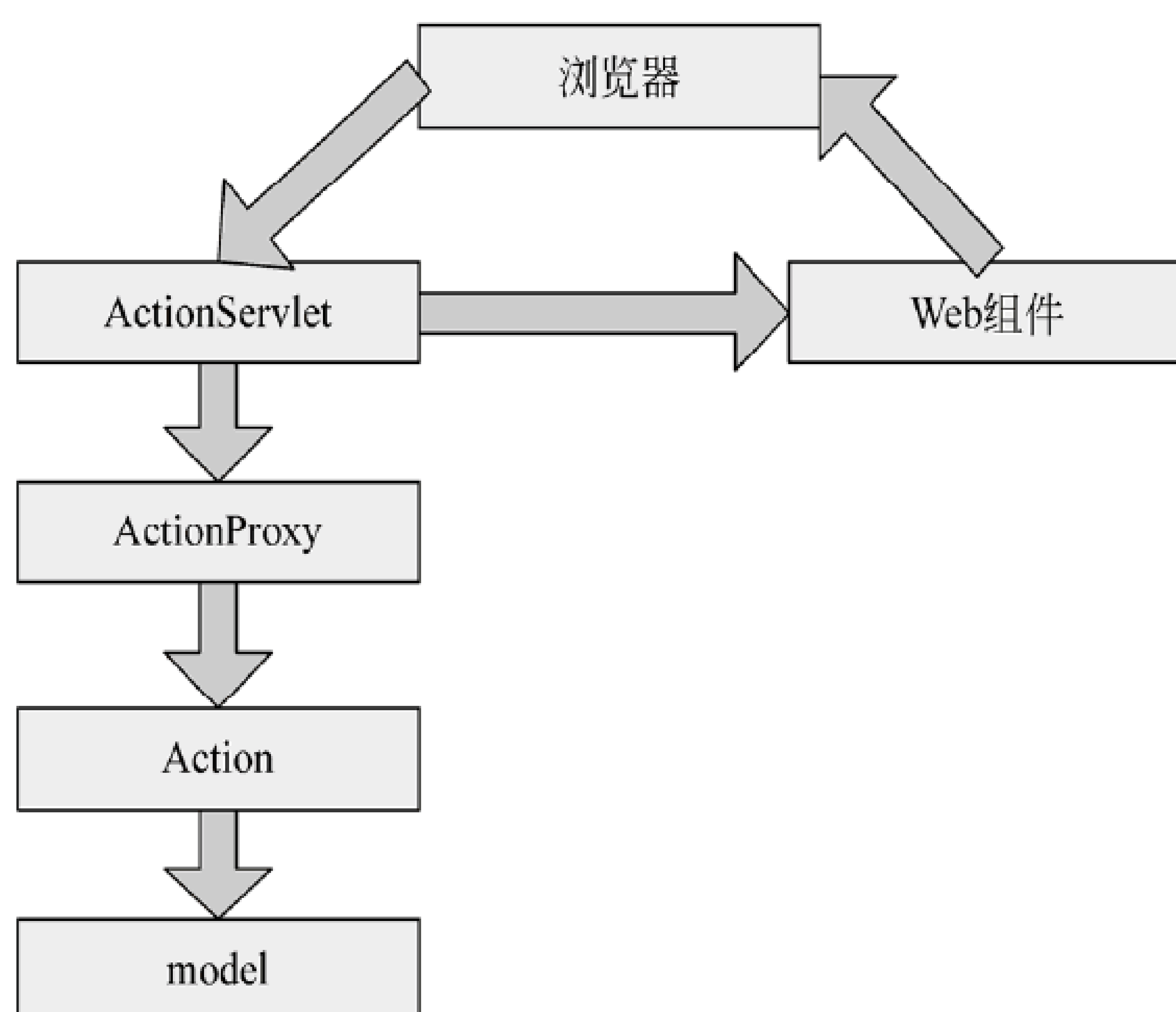


图 3.4　springstruts 项目流程图

注意：由于 Struts 1.x 中经常会创建多个 Action 实例对象，而 Spring 默认会创建一个实例对象，所以在<bean>元素中的 Scope 属性值应该为 prototype，例如:

```
<bean   name="/login"   class="com.cjg.user.actions.LoginAction"   scope=
"prototype">
```

3.1.3　Spring 集成 Hibernate——事务代理功能

Spring 框架与 Hibernate 框架的集成点就在于事物代理方面，就是在 Spring 框架中利用其的 AOP 方式对事务进行声明式配置。为了便于讲解，本节将利用 Spring+Hibernate 框架实现一个具体的应用。首先介绍代码的运行背景，就是把用户信息添加到数据库的同时还要添加日记信息，具体步骤如下。

（1）创建实体类，在 src 目录中创建一个名为 com.cjg.user.model 的包，然后在该包中

创建两个实体类 Log.java 和 User.java。然后在该包中为两个实体创建两个映射文件 Log.hbm.xml 和 User.hbm.xml 文件。

（2）根据实体类 Log.java 和 User.java 生成数据库表，首先把 2.3.4 节的两个工具类复制到包 com.cjg.user.util 中，然后通过运行工具类 ExportDB.java 生成数据库和相应表。

（3）实现业务逻辑，在名为 com.cjg.user.manager 的包中创建名为 LogManager 的接口和实现该接口的类 LogManagerImpl，用来实现关于日记的操作。代码 3.13 和代码 3.14 用来实现保存日记功能。

代码 3.13　添加日记：LogManager.java

```
...
public interface LogManager {
    public void addLog(Log log);                      //添加日记
}
```

代码 3.14　添加日记：LogManagerImpl.java

```
...
public  class  LogManagerImpl  extends  HibernateDaoSupport  implements
LogManager {
    public void addLog(Log log) {
        this.getHibernateTemplate().save(log);        //实现添加日记方法
    }
}
```

接着在包 com.cjg.user.manager 中，创建名为 UserManager 的接口和实现该接口的类 UserManagerImpl，用来实现关于用户的操作。代码 3.15 和代码 3.16 用来实现添加用户的操作。

代码 3.15　添加用户：UserManager.java

```
...
public interface UserManager {
    public void addUser(User user);                   //添加用户信息方法
    throws Exception;
}
```

代码 3.16　添加用户：UserManagerImpl.java

```
...
public  class  UserManagerImpl  extends  HibernateDaoSupport  implements
UserManager {
    private LogManager logManager;                    //创建 logManager 属性
    public void addUser(User user)                    //实现用户添加信息
    throws Exception {
        this.getHibernateTemplate().save(user);
                                              //获取 Session 对象并保存用信息
        Log log = new Log();
        log.setType("123");
        log.setDetail("xxx ");
        log.setTime(new Date());
        logManager.addLog(log);
        throw new Exception();
    }
```

```
    public void setLogManager(LogManager logManager) {
                                                //配置创建 logManager 属性
        this.logManager = logManager;
    }
}
```

【代码解析】

在上述代码 3.14 和代码 3.16 中都使用如下代码：

```
this.getHibernateTemplate().save()
```

其中，this.getHibernateTemplate()就是利用类 HibernateDaoSupport 的方法 getHibernateTemplate()获取包装 Session 对象的模板，然后通过该模板的方法 save()保存信息。

（4）对通用的事务和业务层的类进行配置，代码 3.17 和代码 3.18 分别实现了通用事务的配置和业务层类配置。

代码 3.17　配置事务：applicationContext-common.xml

```
<beans>
    <!-- 配置 sessionFactory -->
    <bean id="sessionFactory" class="org.springframework.orm.hibernate3.
    LocalSessionFactoryBean">
        <property name="configLocation">
            <value>classpath:hibernate.cfg.xml</value>
        </property>
    </bean>
    <!-- 配置事务管理器 -->
    <bean id="transactionManager" class="org.springframework.orm.
    hibernate3.HibernateTransactionManager">
        <property name="sessionFactory">
            <ref bean="sessionFactory"/>
        </property>
    </bean>
    <!-- 配置事务的传播特性 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="add*" propagation="REQUIRED"/>
            <tx:method name="*" read-only="true"/>
        </tx:attributes>
    </tx:advice>
    <!--配置相关类的相关方法参与事务 -->
    <aop:config>
        <aop:pointcut id="allManagerMethod" expression="execution(* com.
        cjg.user.manager.*.*(..))"/>
        <aop:advisor pointcut-ref="allManagerMethod" advice-ref=
        "txAdvice"/>
    </aop:config>
</beans>
```

代码 3.18　配置业务层类：applicationContext-beans.xml

```
<beans >
    <!--对用户类进行配置 -->
    <bean id="userManager" class="com.bjsxt.usermgr.manager.
    UserManagerImpl">
        <property name="sessionFactory" ref="sessionFactory"/>
```

```
        <property name="logManager" ref="logManager"/>
    </bean>
    <!--对日记类进行配置 -->
    <bean id="logManager" class="com.bjsxt.usermgr.manager.Log
    ManagerImpl">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>
</beans>
```

（5）创建测试类，在名为 com.cjg.user.test 的包中创建一个类 Test.java。代码 3.19 实现了向用户信息和日记信息中添加数据。

代码 3.19　测试类：Test.java

```
public class Test {
    public static void main(String[] args) {
        User user = new User();                    //创建 user 对象
        user.setName("12");                        //设置 user 对象的属性
        //获取 BeanFactory 对象
        BeanFactory factory = new ClassPathXmlApplicationContext
        ("applicationContext-*.xml");
        //创建 userManager 对象
        UserManager userManager = (UserManager)factory.getBean
        ("userManager");
        try {
            userManager.addUser(user);             //实现添加用户功能
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

（6）运行代码，当编译完 Test.java 代码后，查看数据库时就会得到用户和日记的数据分别如图 3.5 和图 3.6 所示。

	id	name
□	1	12
*	(NULL)	(NULL)

图 3.5　用户

	id	type	detail	time
□	1	123	xxx	2009-08-01 17:35:38
*	(NULL)	(NULL)	(NULL)	(NULL)

图 3.6　日记表格

当使用集成框架来开发大型的项目时候，一般会分成 4 层（如图 3.7 所示），各层实现的功能如下。

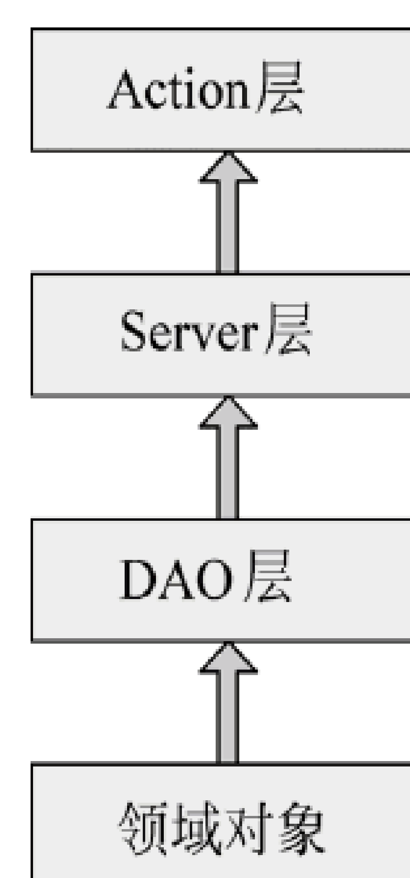


图 3.7　4 层体系

- POJO 层：该层为最低层，主要是通过 Hibernate 框架的映射技术针对数据库中的每个表格创建相对应的 POJO 对象，这些对象是非常关键的，将来会在各个层进行传递。
- DAO 层：该层实际上也是通过 Hibernate 框架针对数据库中每个表格实现基本的增、删、改、查（CRUD）、分页等功能。在具体实现时，一般会先创建该层的接口，然后创建实现各种功能的类，这些类不会直接使用 Hibernate 框架，而是通过继承该层接口类来实现。
- Server 层：该层主要是通过实现事务的添加等操作来实现业务逻辑，在具体实现时，

该层的一个方法会调用 DAO 层的多个方法。

- ❑ Action 层：该层通过调用 Server 层的方法来处理 Web 请求，这些方法会由 Sping 框架来管理和注入。

每当涉及数据操作时，就会使用事务处理，为什么 DAO 层与 Manager 层都涉及数据处理却把事务处理方面放在了 Manager 层？因为在 DAO 层只是针对数据库中的每个表格进行操作，而事务处理却是针对数据库中的多个表格来实现相对应的功能，所以如果在 DAO 层实现事务，这些事务就会不完整。

注意：一般会用包来包含各层的相关类，在创建包时要注意包的命名规范，一般以 com 开始，接着为公司或个人的名字，然后为项目的名称，最后为该类属性那一层。

3.2　实现 SSH 三种框架环境集成

在 SSH（Struts 1.x+Spring+Hibernate）集成框架中，Hibernate 框架用来实现持久层数据库操作，Struts 1.x 框架用来实现控制跳转和显示，而 Spring 框架用来实现事务管理、接口注入等处理以提高代码质量和对象的独立性。

本节将介绍如何实现 Struts 1.x 框架、Spring 框架和 Hibernate 框架三者的集成。由于在讲解 Spring+Struts 1.x 和 Spring+Hibernate 集成时，都是实现了一个具体的实例，所以本节将不通过具体的实例来讲解如何实现 Struts 1.x+Spring+Hibernate 框架的集成，而只是讲解一下三者集成环境的实现过程。

3.2.1　配置数据库字符集体

在 SSH 集成的环境中，必定会使用到数据库。为了能够在数据库中既能存储英文字符又能存储中文字符，必须先配置数据库的字符集。针对 MySQL 数据库字符集可以进行如下配置。

（1）首先通过“开始”|“所有程序”|MySQL|MySQL Server 5.0|MySQL Server Instance Config Wizard 命令打开如图 3.8 所示的欢迎对话框。

（2）单击 Next 按钮进入如图 3.9 所示的 MySQL Server 5.0 Instance Configuration（安装配置）对话框。在该对话框中选中 Reconfigure Instance 选项，就会对 MySQL 数据库进行重新配置。

（3）接着连续单击 7 次 Next 按钮（每一页保持默认配置就可以），就会进入如图 3.10 所示的选择字符集对话框。在该对话框中默认会选择 Standard Character Set 选项，即 ISO-8859-1(Latin 1)字符集，该字符集仅仅只允许使用西方字符。在具体开发中，推崇选择 Best Support For Multilinqualism 选项，即 UTF-8 字符集。之所以选择该字符集，因为 UTF-8 字符集包含了当今世界上所有的语言，还有就是如果页面中不仅需要使用英文和中文而且还需要使用日文，那只能使用 UTF-8 字符集。

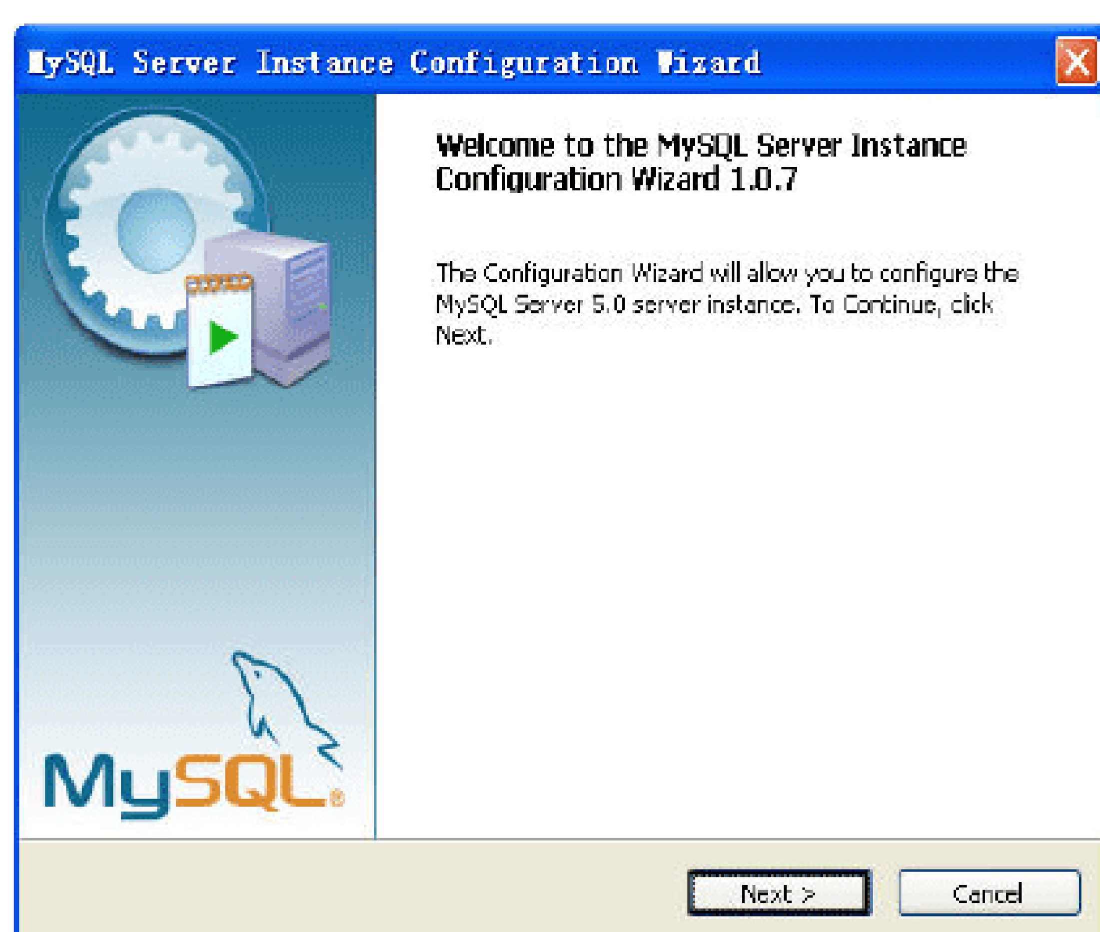


图 3.8　欢迎对话框

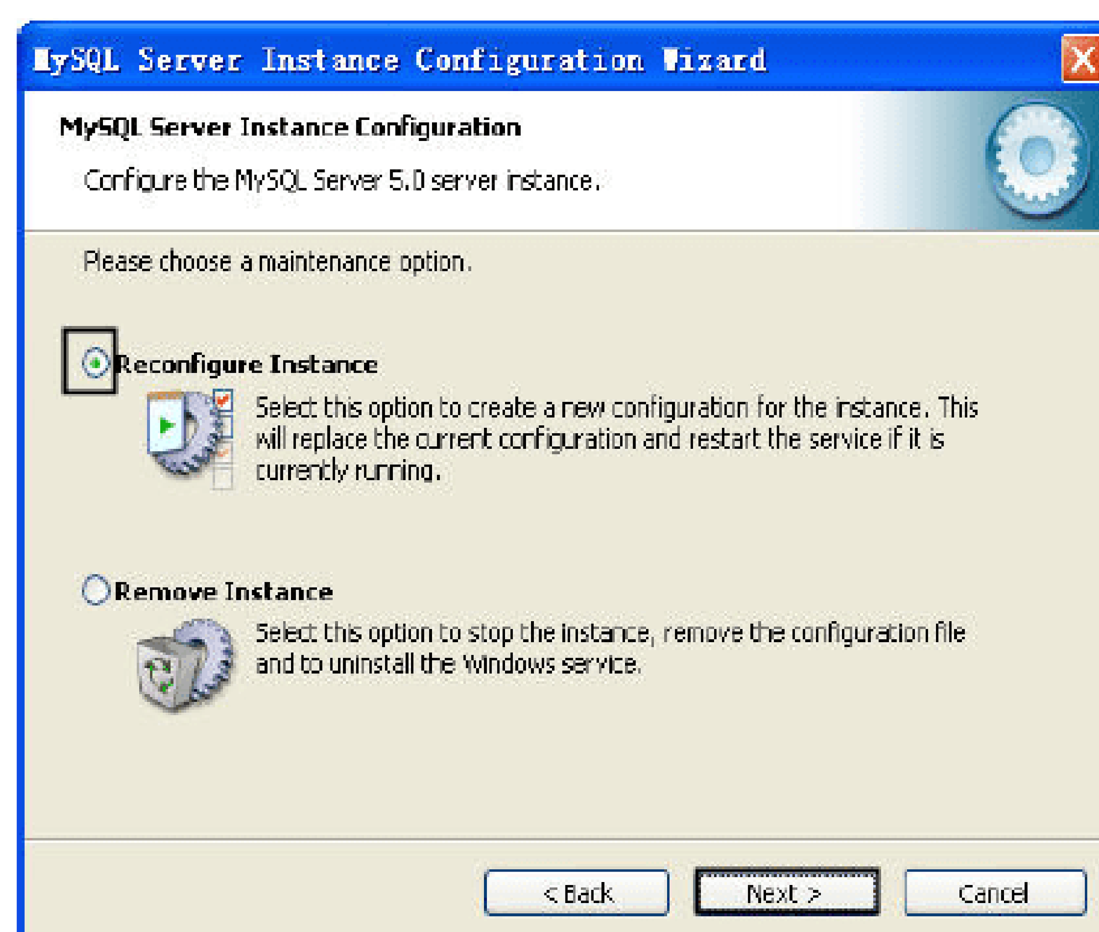


图 3.9　安装配置对话框

注意：如果选择了 GBK 字符集，那么在页面中只能使用中文。对该项的设置主要是解决从数据库中取出来的字符为乱码的问题。

（4）接着单击 Next 按钮就会弹出执行配置对话框，单击 Execute 按钮过一段时间就会进入执行完毕对话框，如图 3.11 所示。在该对话框中单击 Finish 按钮完成对字符集的配置。

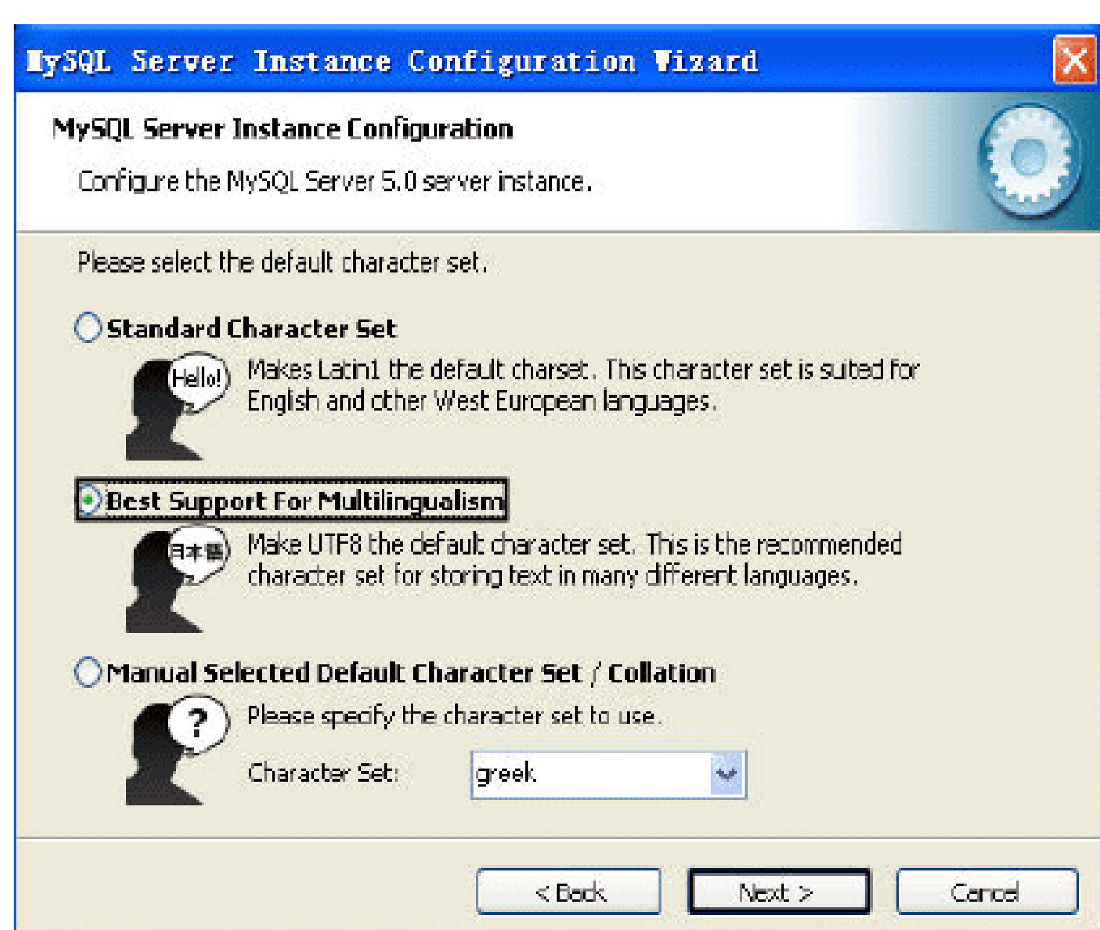


图 3.10　选择字符集对话框

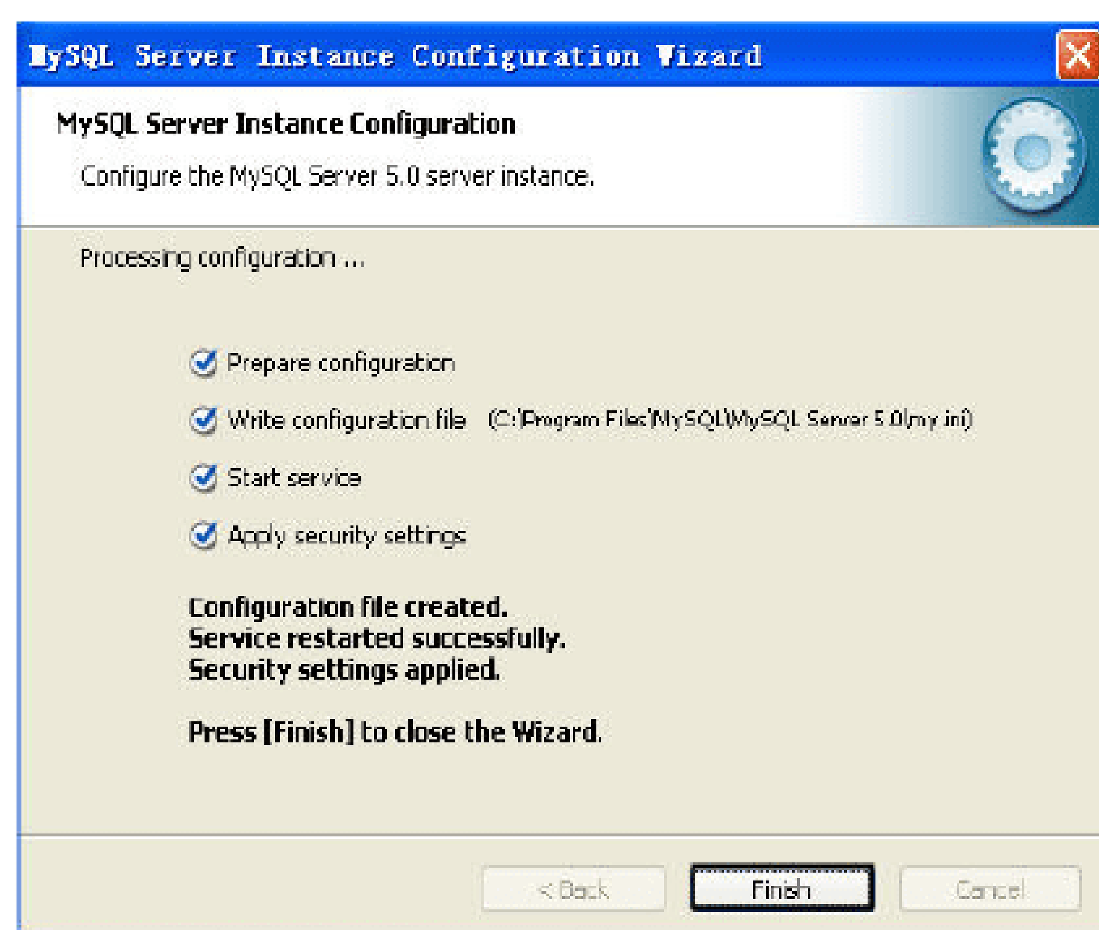


图 3.11　执行配置对话框

3.2.2　集成 Hibernate

开发环境 MyEclipse 对 Hibernate 框架的集成提供了全方位的支持，为了使其能够与其他框架集成，必须做一些必要的设置。如何在 MyEclipse 开发环境中实现 Hibernate 框架与其他框架的集成环境呢？详细步骤如下。

（1）新建一个名为 strutsspringhibernate 的 Web Project。

（2）为项目增加 Hibernate 相关类库与文件，在 Add Hibernate Capabilities（添加 Hibernate 框架）向导的第 1 页要注意选择 Hibernate 的版本为 3.2，把 JAR Library Installation（安装 jar 文件）选项修改成如图 3.12 所示。这是因为当选择上面的选项时，关于 Hibernate 的 jar 文件会存放到项目的 classpath 目录下。对于这种情况，在 MyEclipse 开发环境中编译和运行不会出现问题。可是如果部署到 Tomcat 服务器上就会出现问题。

如果想查看关于 Hibernate3.2 的 jar 包，可以通过单击图 3.12 中的 View and edit libraries 链接，弹出如图 3.13 所示的对话框，在其中就可以查看相关的 jar 包。

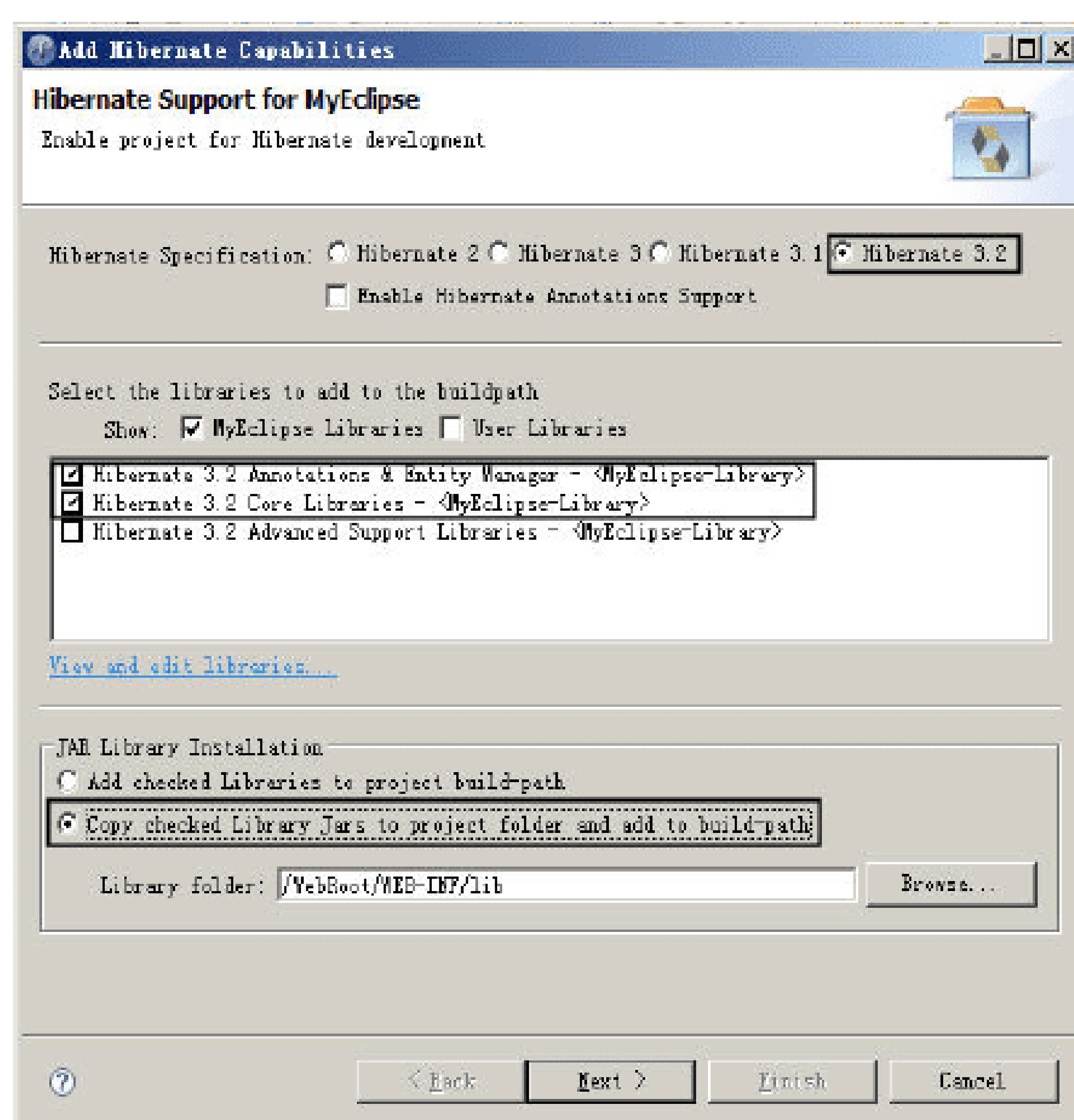


图 3.12　添加 Hibernate 相关文件

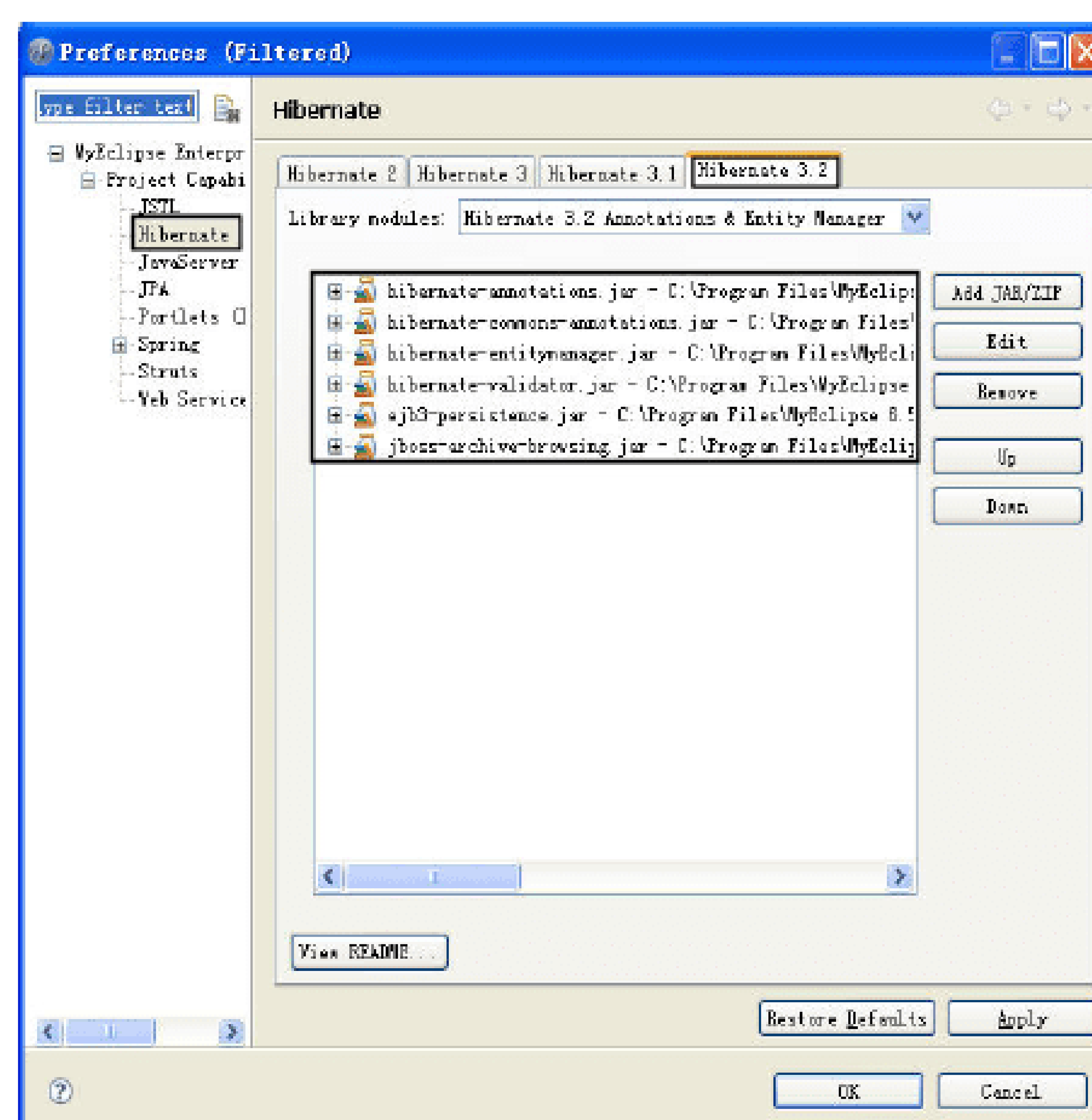


图 3.13　查看相关 jar 文件

（3）单击 Next 按钮进入向导的第 2 页，保持默认选项就可以。接着再单击 Next 按钮进入向导的第 3 页（如图 3.14 所示），在这一页需要指定数据库连接的相关信息。由于在该项目中是由 Spring 框架来管理数据库方面的信息，所以需要取消选中 Specify database connection details 复选框，不做相关配置。接着再次单击 Next 按钮进入向导的第 4 页，在这一页需要指定是否创建 SessionFactory。由于在该项目中是由 Spring 框架来管理 SessionFactory 类，在该页中去掉复选框 Create SessionFactory class 的选中状态，最后单击 Finish 按钮完成添加 Hibernate 框架到项目的过程。

完成支持 Hibernate 框架开发的环境后，其目录结构如图 3.15 所示，至此该项目已经支持了 Hibernate 框架技术。

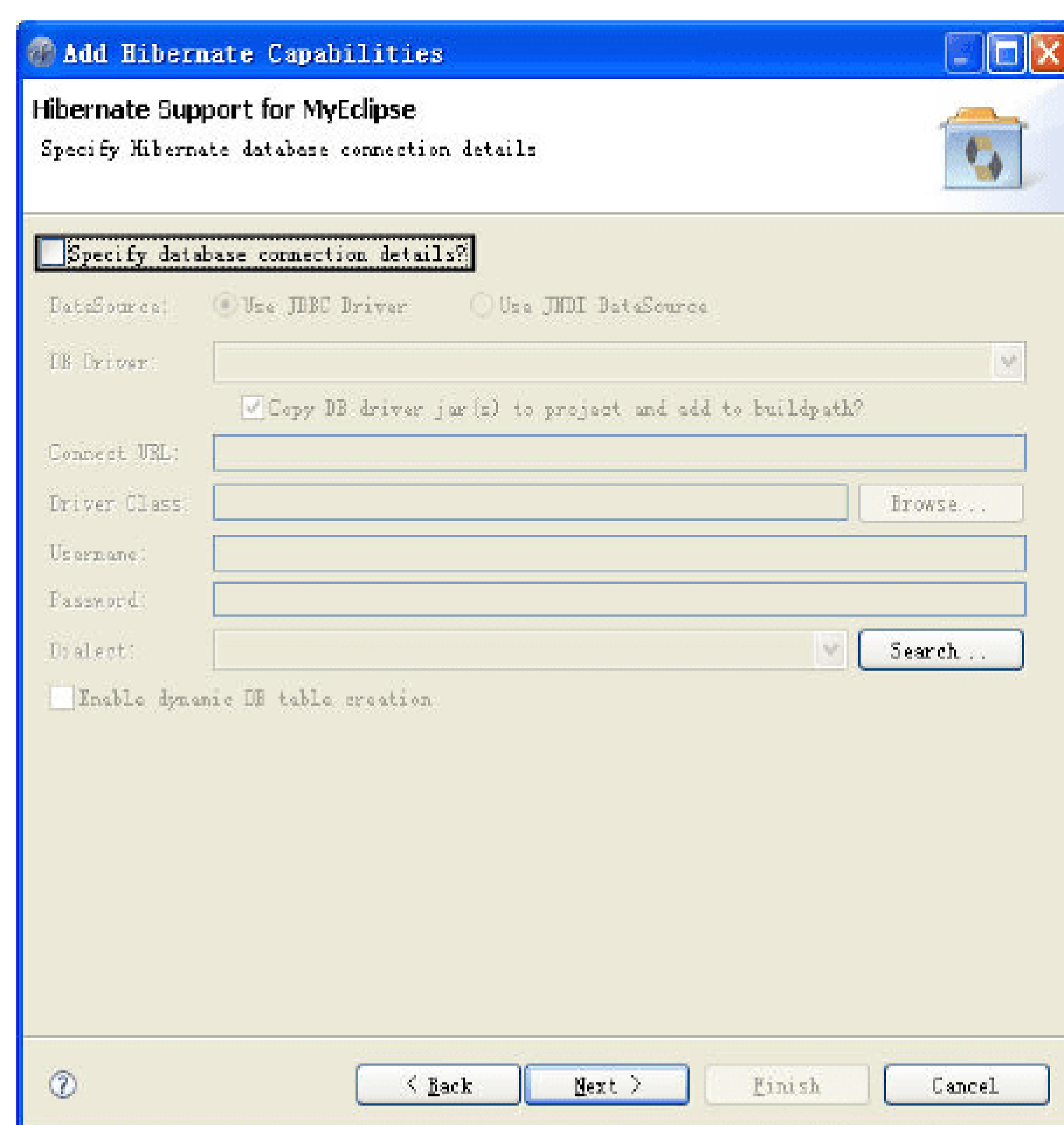


图 3.14　配置数据库连接

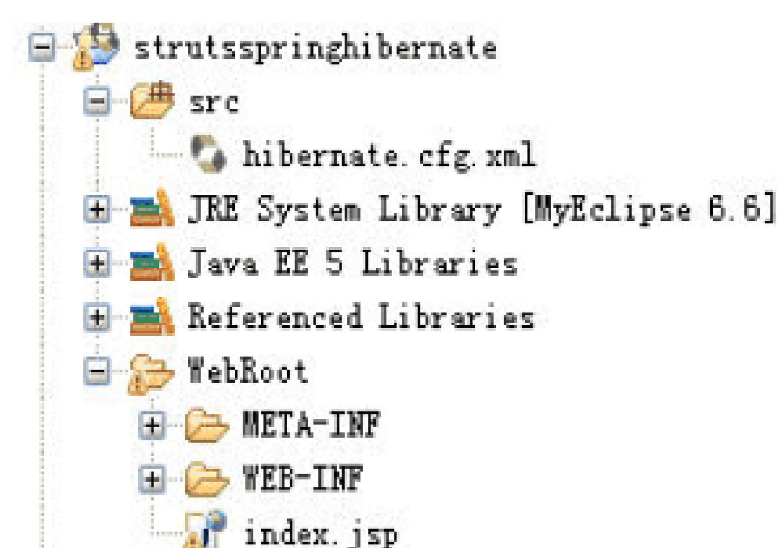


图 3.15　目录结构

3.2.3　集成 Spring 框架

在连接数据库方面，Spring 框架支持 Hibernate、JRA 等数据库访问技术的集成开发。

在与数据库连接的方面，Spring 框架主要提供了调用类和事务管理功能。开发环境 MyEclipse 对 Spring 框架与 Hibernate 框架的集成提供了全方位的支持。接着 3.2.2 节项目继续增加对 Spring 框架的支持，具体步骤如下。

（1）通过向导使当前的项目支持 Spring 框架应用。在添加 Spring 框架向导的第 1 页除了要注意把 JAR Library Installation 选项修改成如图 3.16 所示，还需要注意选择 Spring 框架的版本为 2.0，这是因为在该项目中只需要 Spring 2.0 的 jar 文件就可以。最后在 Select the libraries to add to the buildpath 的（选择添加到类路径的类库）下侧，选中 Spring 2.0 AOP Libraries（关于 AOP 方面的 jar 文件）、Spring 2.0 Core Libraries（Spring 框架的核心 jar 文件）、Spring 2.0 Persistence Core Libraries（关于持久化方面的 jar 包）、Spring 2.0 Persistence JDBC Libraries（关于持久化方面的 jar 包）和 Spring 2.0 Web Libraries（关于 Web 开发方面的 jar 包）5 个类型。

如果想查看关于 Spring 2.0 的 jar 包，可以通过单击图 3.16 中的 View and edit libraries 链接，这时就会弹出如图 3.17 所示的对话框，在该图中就可以查看具体的 jar 包。

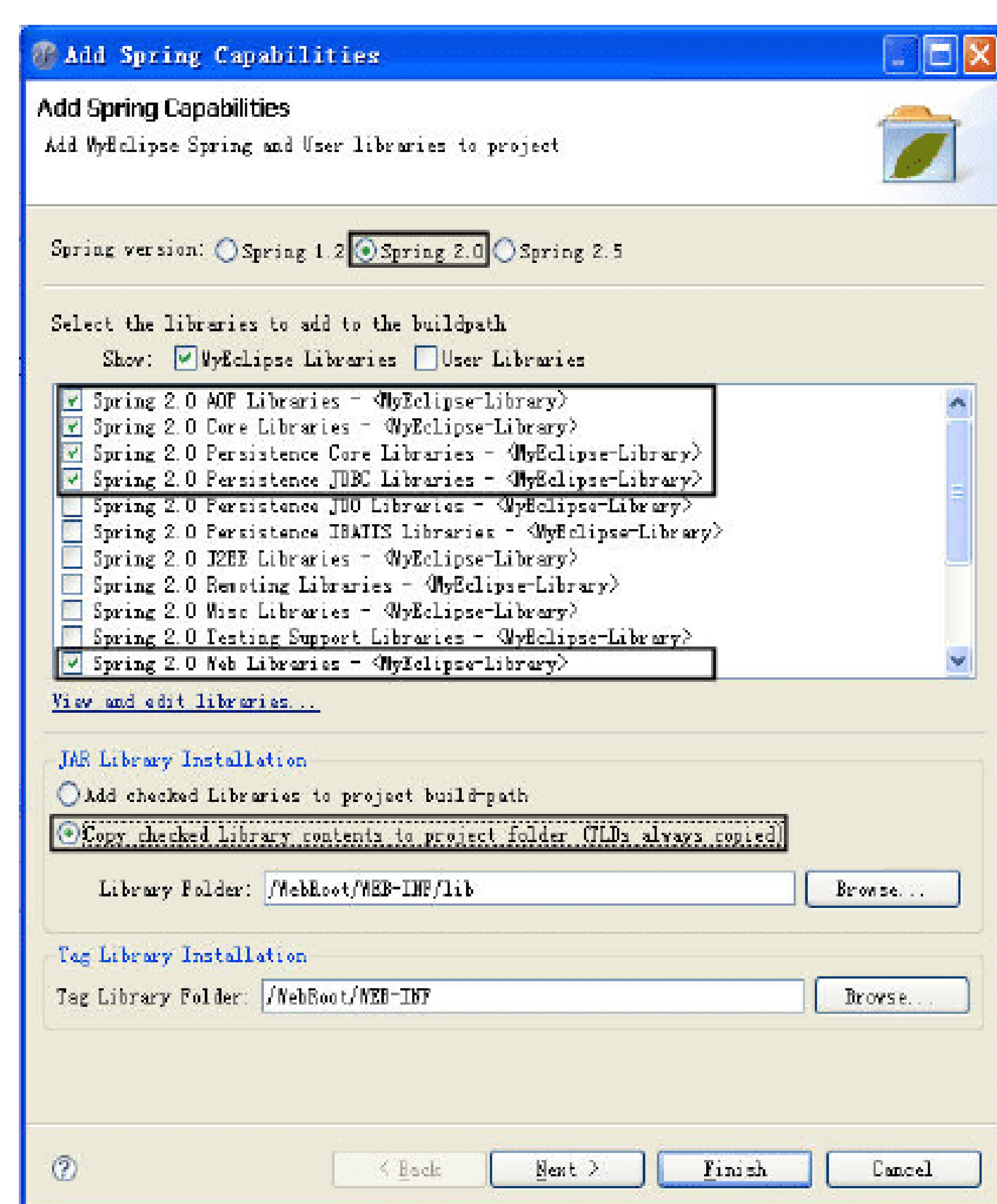


图 3.16　添加 Spring 框架相关文件

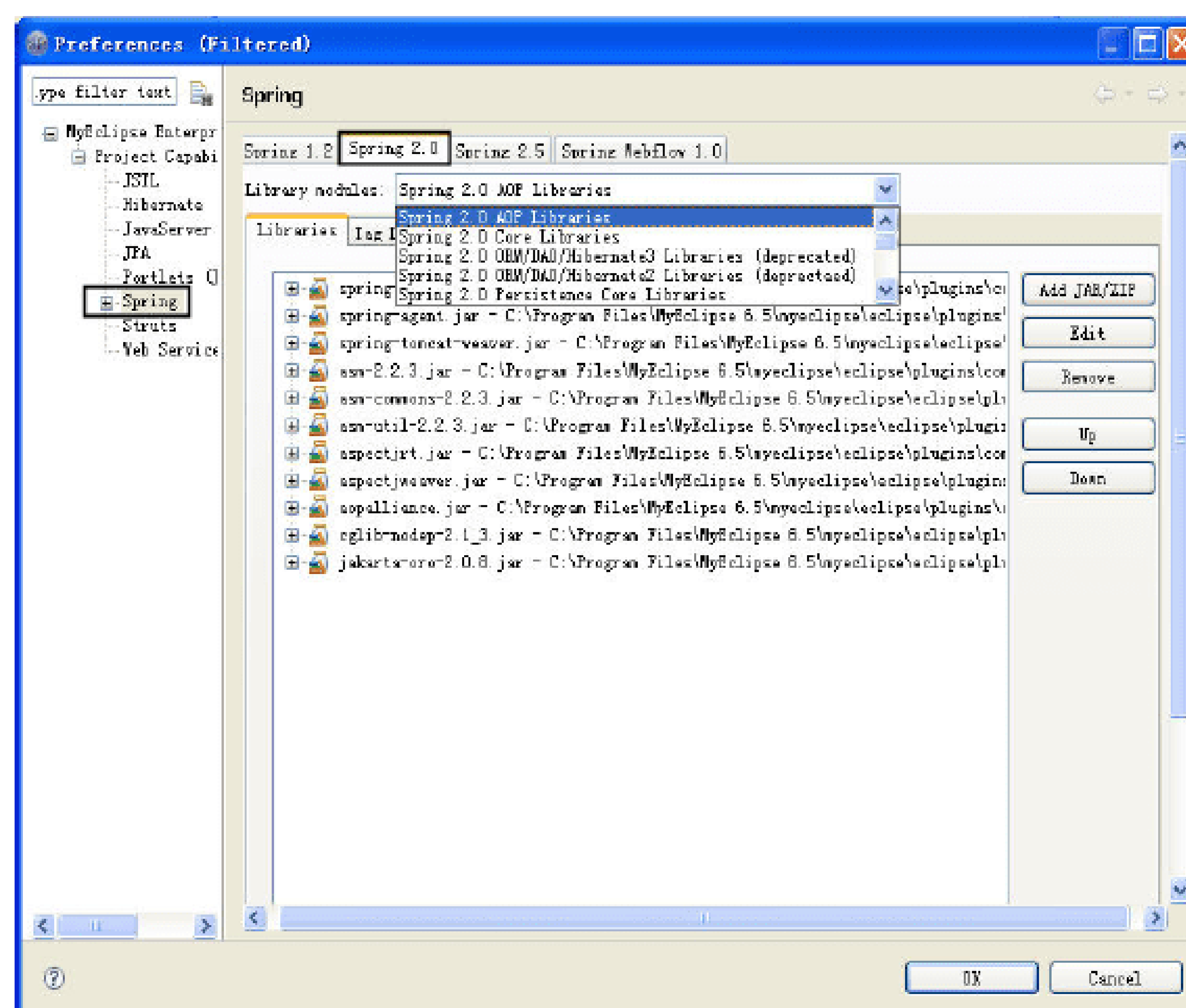


图 3.17　查看相关 jar 文件

（2）单击 Next 按钮进入向导的第 2 页，该页用来设置 Spring 框架配置文件。首先选择 Specify new or existing 复选框，接着通过 Browse 按钮来修改 Spring 框架配置文件的存放位置，如图 3.18 所示。

（3）接着单击 Next 按钮就进入第 3 页，该页都保持默认选项就可以，最后单击 Finish 按钮完成添加 Spring 框架到项目的过程。注意该对话框的第 3 页是专门设置 Spring 框架与 Hibernate 框架集成信息的，在该页中其实就是配置 Spring 框架下的 LocalSessionFactory 对象，该对象是关于 Hibernate 配置文件的信息，如图 3.19 所示。

完成支持 Spring 框架与 Hibernate 框架开发的环境后，其目录结构如图 3.20 所示。至此该项目已经支持了 Spring 框架与 Hibernate 框架集成技术。

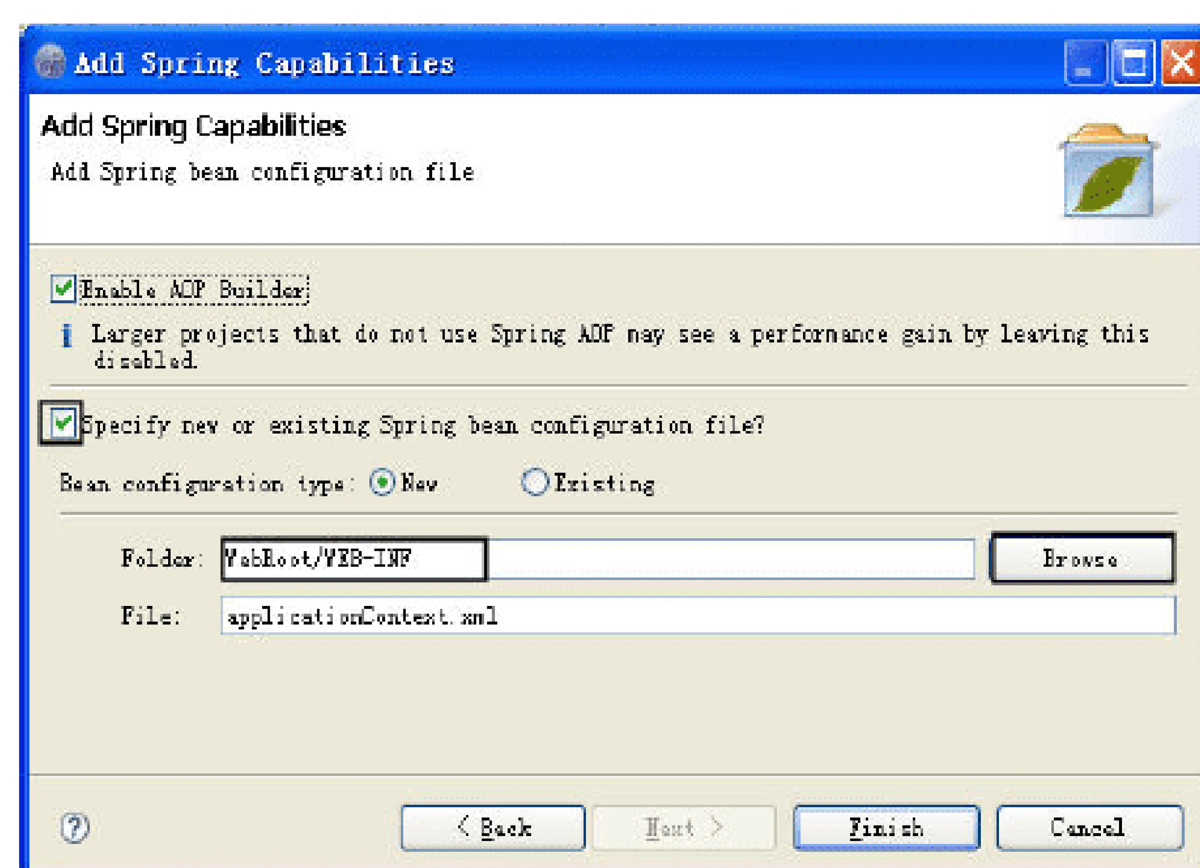


图 3.18　设置 Spring 框架的配置文件

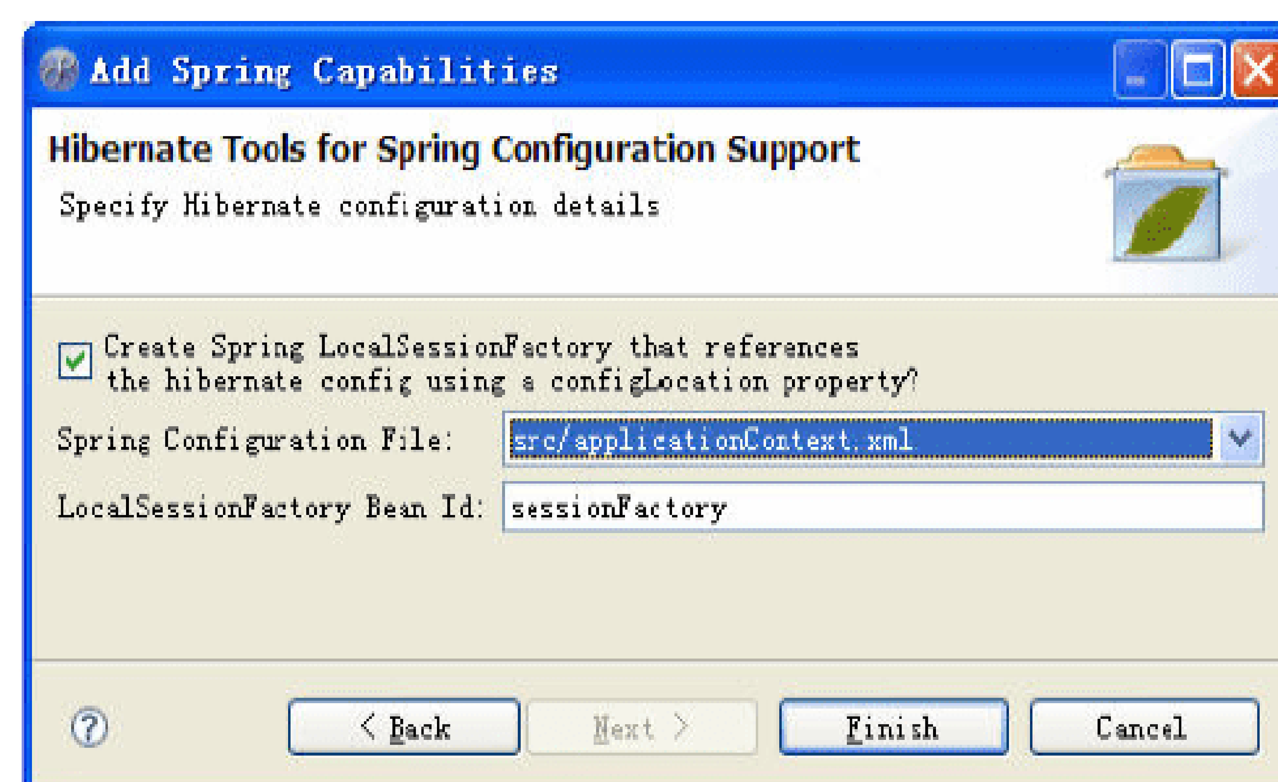


图 3.19　添加 Spring 配置的 Hibernate 工具

3.2.4　集成 Struts 1.x 框架

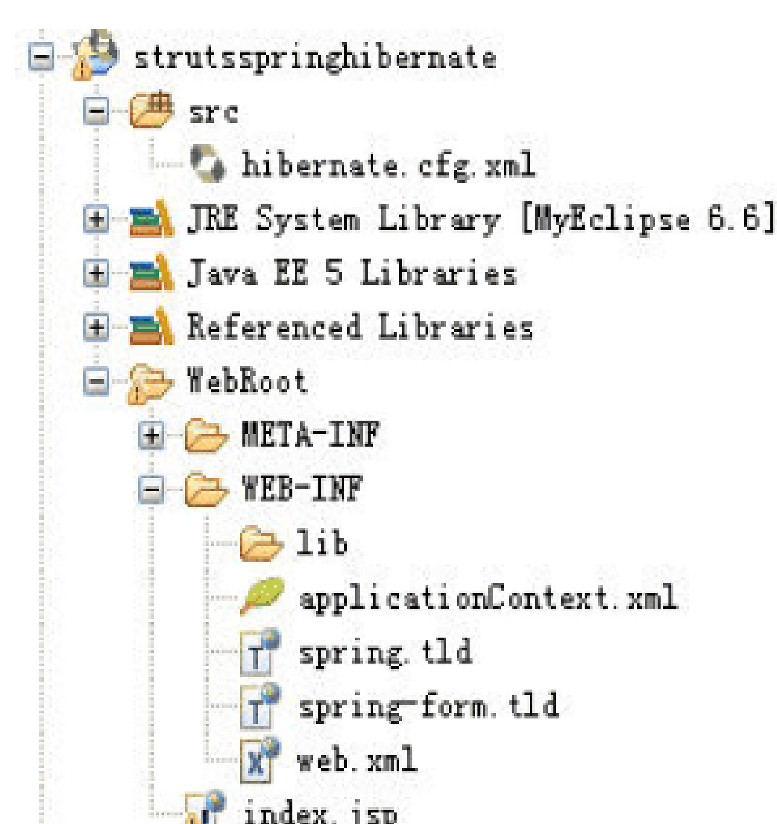


图 3.20　目录结构

开发环境 MyEclipse 对 Struts 1.x 框架的集成提供了全方位的支持，为了使其能够与其他框架集成，必须做一些必要的设置。如何在 MyEclipse 开发环境中实现 Struts 1.x 框架与其他框架的集成环境呢？接着 3.2.3 节项目继续增加对 Struts 1.x 框架的支持，具体步骤如下。

通过向导使当前的项目支持 Struts 1.x 框架应用。在添加 Struts 1.x 框架向导的第 1 页注意选择 Struts 框架的版本为 1.2，这是因为在该项目中只需要 Struts 1.2 的 jar 文件就可以。接着修改 Base package for new classes 和 Default application resources 两个选项的包名称，最后的配置结果如图 3.21 所示。

如果想查看关于 Struts 1.2 的 jar 文件，可以通过单击图 3.21 中 View and edit libraries 链接，就会弹出如图 3.22 所示的对话框，在该图中就可以查看具体的 jar 包。

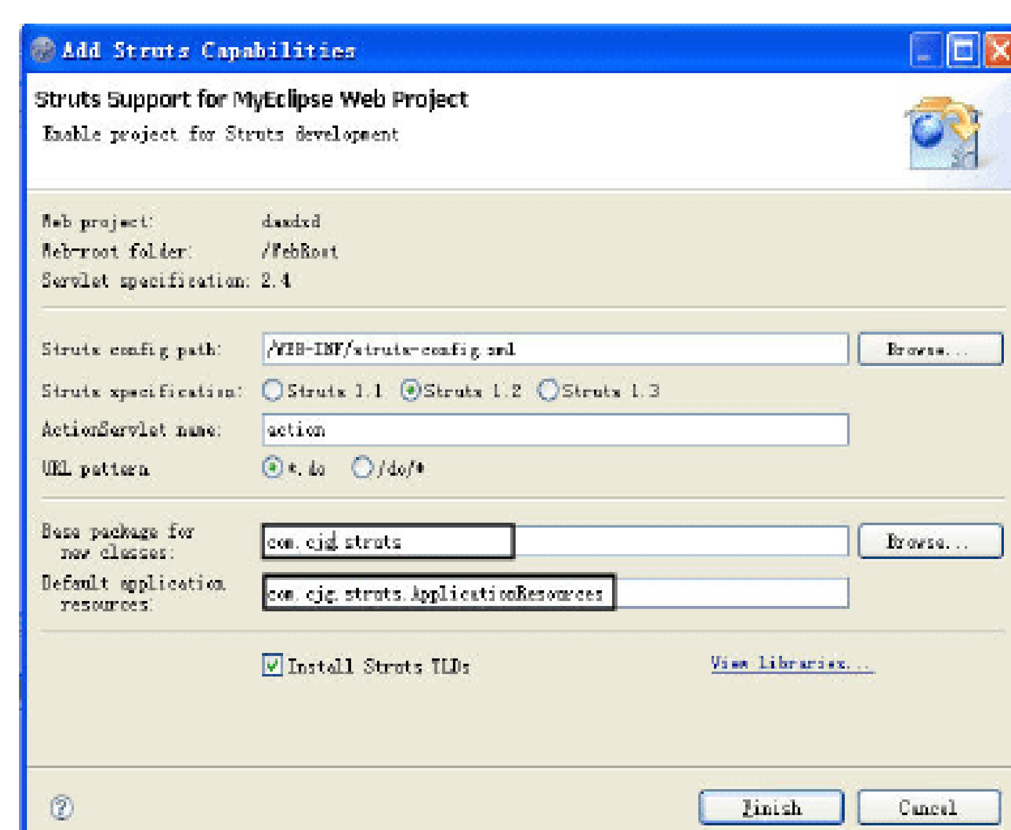


图 3.21　添加 Struts 框架相关文件

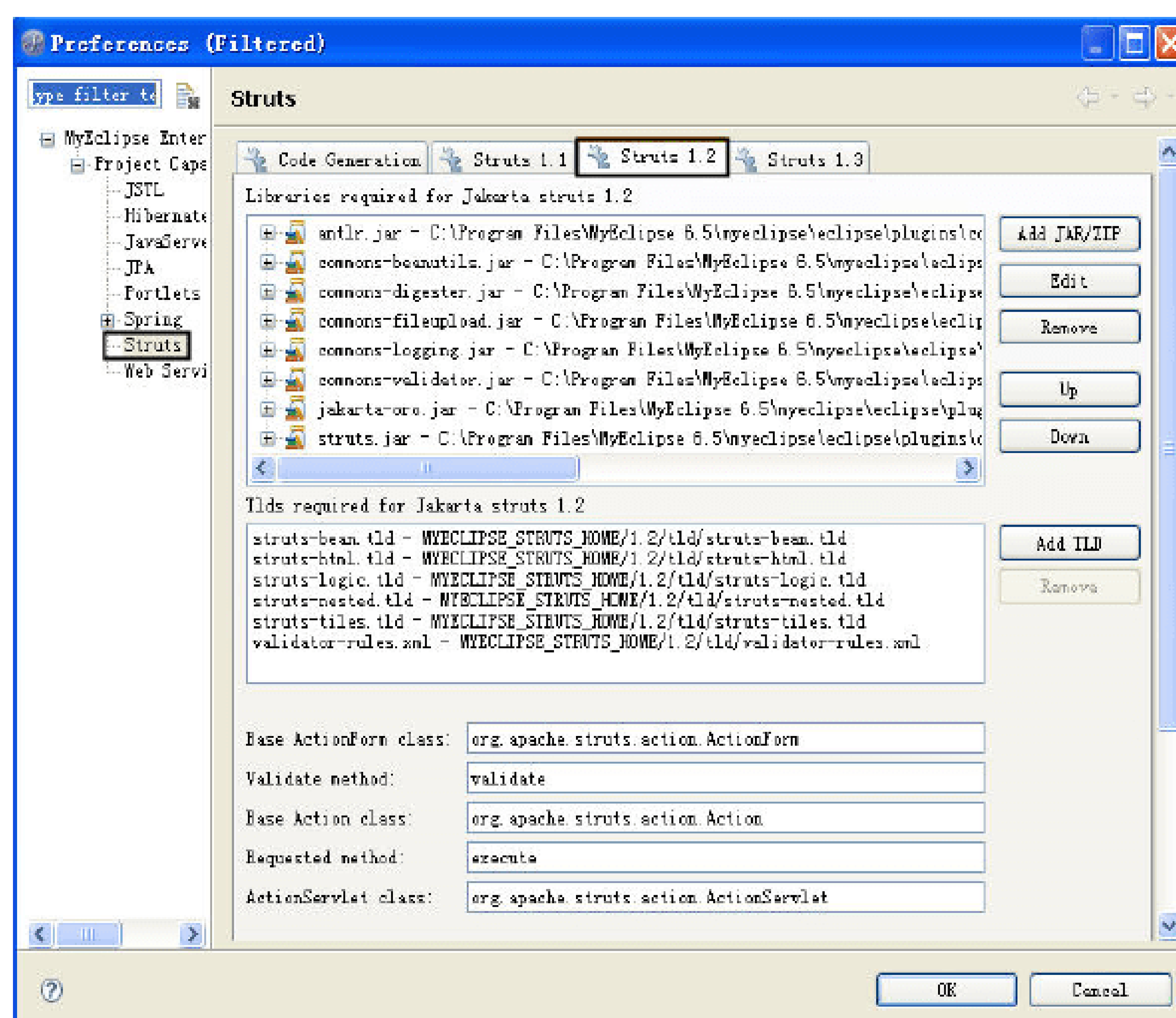


图 3.22　查看相关 jar 文件

完成支持 Struts 1.x 框架、Spring 框架和 Hibernate 框架集成开发的环境后，其目录结构如图 3.23 所示。至此，该项目已经具备 Struts 1.x 框架、Spring 框架和 Hibernate 框架的技术支持。

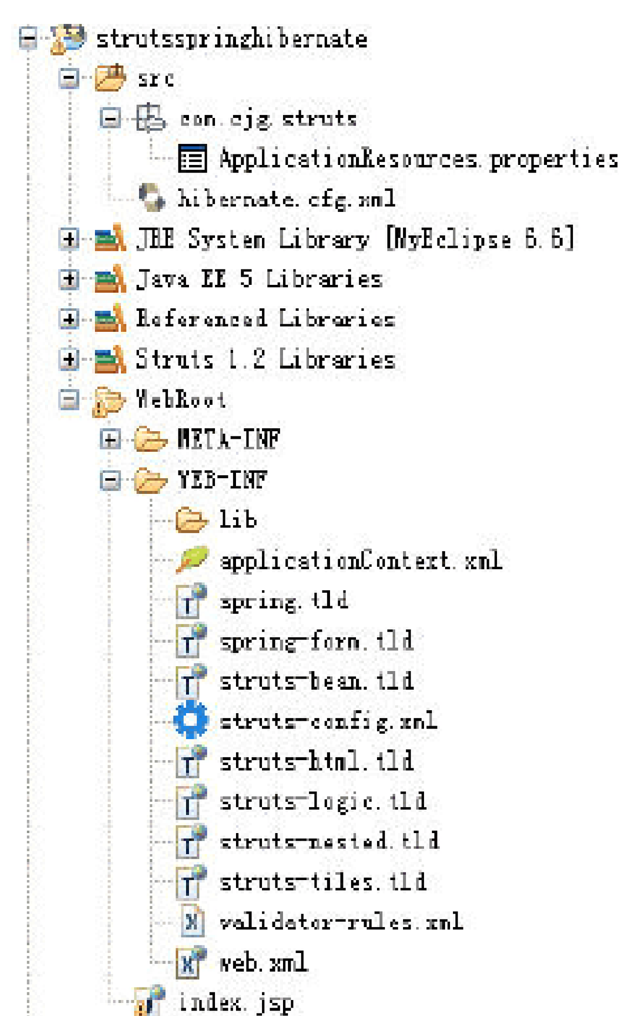


图 3.23　目录结构

3.3　实现 Spring 与 Struts 2.x 集成

通过上面几节的学习可以发现，如果想实现 Spring 框架和 Struts 1.x 框架的集成，可以通过两种方案来实现。但是如果想实现 Spring 框架和 Struts 2.x 框架的集成，是否还可以通过上述两种方案来实现？答案肯定是否定，这是因为 Struts 2.x 框架主要是通过插件来实现对其他框架的支持。

3.3.1　关于 Spring 框架的插件

Spring 框架与 Struts 2.x 框架的集成过程非常简单，因为 Struts 2.x 框架中已经提供了关于 Spring 框架的插件。对于 Struts 2.x 框架中所有的插件，其实就是一些名称不同的 jar 文件，一般以“Struts2-框架名-plugin-版本号”的方式命名。同理，关于 Spring 框架的插件也不例外。

下载了关于 Struts 2.0.11 的完整开发包后，就可以在该开发包的 lib 文件夹中找到 struts2-spring-plugin-2.0.11.jar 这个插件。当对该 jar 文件解压后就会发现里面包含了一个名为 struts-plugin.xml 的 XML 文件，如图 3.24 所示。通过修改该文件，可以实现一些特殊功能：定义新包和新的结果类型、覆盖 Struts 2.x 框架中的常量、自定义拦截器、改变默认拦截器的引用和引入扩展点的实现类。

注意：对于 struts-plugin.xml 和 struts.xml 配置文件，Struts 2.x 框架首先加载的是 struts-plugin.xml 配置文件，其次才是 struts.xml 配置文件。

如何在 Spring 和 Struts 2.x 框架集成的项目中安装关于 Spring 框架的插件呢？一般分为两个步骤。

（1）首先将关于 Spring 框架的插件复制到项目的 classpath 下，即先复制插件的 jar 文件到项目名称/WebRoot/lib 目录下，然后右击该文件，在弹出的快捷菜单中（如图 3.25 所示）选择 Build Path|Add to Build Path 命令就会把该文件转移到 classpath 下。

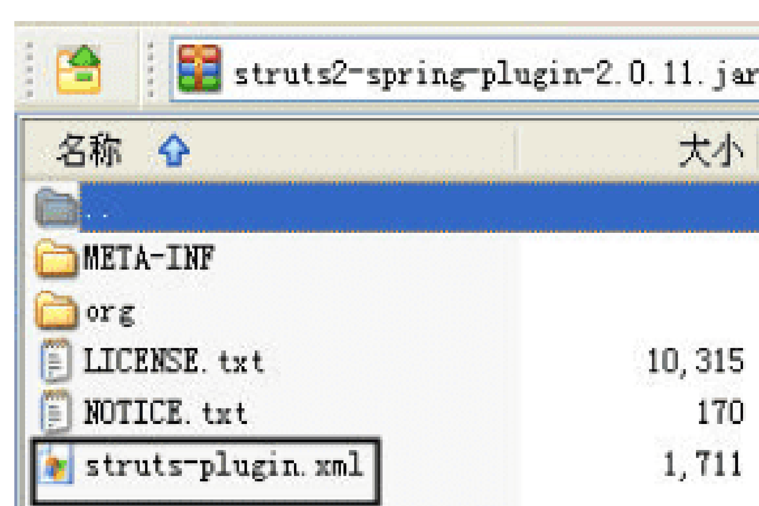


图 3.24　struts-plugin 文件

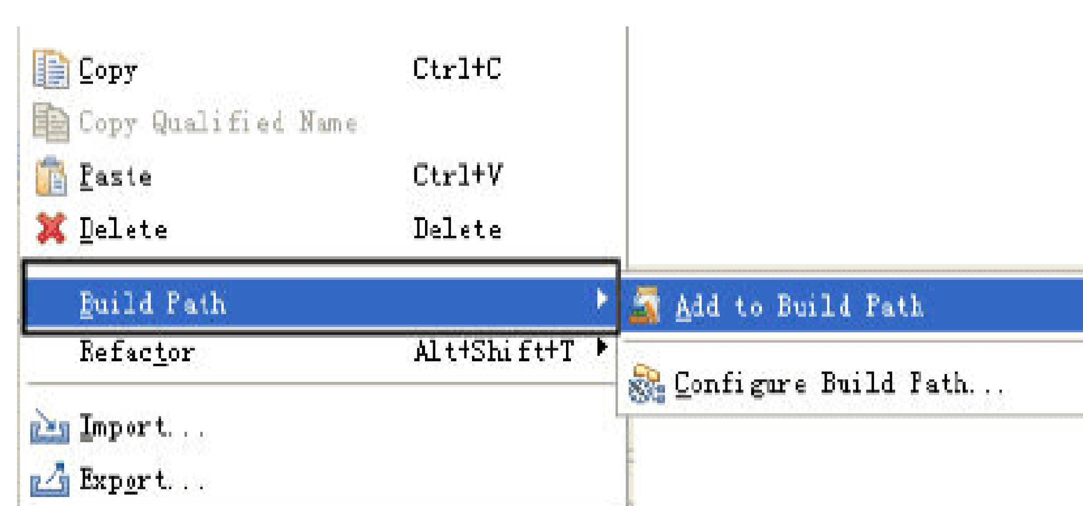


图 3.25　添加 jar 文件

（2）接着修改 web.xml 配置文件，使该项目能够监听 Spring 框架，没有配置 Spring 框架时的内容如代码 3.20 所示，配置 Spring 框架后的内容如代码 3.21 所示。

代码 3.20　没配置 Spring 的文件：web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
...
    <filter>                                <!--配置 struts 2 过滤器-->
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <filter-mapping>                        <!--配置 struts 2 过滤器映射-->
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

代码 3.21　配置 Spring 的文件：web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
...
    <filter>                                <!--配置 struts 2 过滤器-->
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <filter-mapping>                        <!--配置 struts 2 过滤器映射-->
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <listener>                              <!--关于 Spring 框架的监听-->
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
...
</web-app>
```

注意：在没有配置 Spring 框架时，Struts 2 中的 Action 都是在 struts.xml 中声明，但是当使用 Spring 框架后，这些 Action 对象的实例化过程就会由该框架实现。

3.3.2　Spring 与 Struts 2.x 框架集成

为了便于讲解，本节将利用 Struts 2.x 和 Spring 框架实现一个具体实例来介绍两者集成的详细过程。首先介绍代码的运行背景，即简单的登录系统，具体步骤如下。

（1）首先新建一个名为 struts2spring 的 Web Project 项目，然后引入 Struts 2.x 框架的相关 jar 文件，对于 Struts 2.x 框架需要这些 jar 文件：struts2-core-2.0.11.jar、xwork-2.0.4.jar、ognl-2.6.11.jar、freemarker-2.3.8.jar 和 commons-logging-1.0.4.jar。接着由于该框架要与 Spring 框架集成，所以还需要 struts2-spring-plugin-2.0.8.jar 文件。最后通过 MyEclipse 开发环境的向导添加关于 Spring 框架的支持，这时该项目的目录如图 3.26 所示。

（2）在 web.xml 中添加关于 Struts 2.x 和 Spring 框架的监听器。在 web.xml 文件中一定要注意各个监听器的顺序。

（3）建立一个用来实现登录的页面，该页面的具体内容如代码 3.22 所示。为了接受该页面发出的请求，还必须创建一个名为 LoginService.java 类，用来对请求的值进行处理，具体内容如代码 3.23 所示。最后还应该创建一个 Struts 2.x 框架的 Action 类，通过不同的处理结果实现页面的跳转，具体内容如代码 3.24 所示。

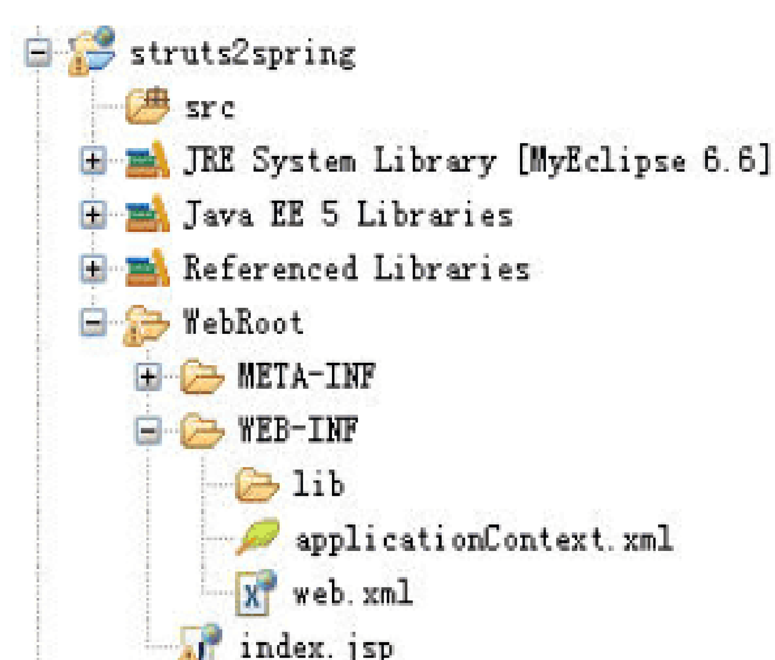


图 3.26　项目目录

代码 3.22　登录页面：login.jsp

```
<!--page 和 taglib 指令-->
<%@ page contentType="text/html; charset=GBK"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
...
    <body>
            <font color="red"> <s:actionerror /> </font><!--输出出错信息-->
            <s:form action="login">                        <!--form 的处理类-->
                            <label>                        <!--用户输入框-->
                                <s:textfield name="username" label="用户
                                名" />
                            </label>
                            <label>                        <!--密码输入框-->
                                <s:textfield name="password" label="密
                                码" />
                            </label>
                            <label>                        <!--提交和重置按钮-->
                                <s:submit value="提交" />
                                <s:reset value="重置" />
                            </label>
            </s:form>
    </body>
...
```

代码 3.23　处理请求：LoginService.java

```
...
public class LoginService {                                    //登录类
    public Integer login(String username, String password) {   //登录方法
```

```
        if (username.equals("cjgong") && password.equals("123456")) {
                                                                //判断参数
            return 1;
        } else
            return 0;
    }
}
```

代码 3.24　处理请求的 Action 类：Login.java

```
...
public class Login extends ActionSupport {
    //编写各种属性
    private String username;
    private String password;
    private LoginService service;
    public String execute() throws Exception {          //编写 execute()方法
        Integer result = service.login(username, password);
        if (result == 1)
            return SUCCESS;
        else {
            return ERROR;
        }
    }
    //配置相关属性的 set()和 get()方法
...
}
```

（4）由于在该项目中是由 Spring 框架来完成对各种类的实例化，所以需要在 applicationContext.xml 文件中对这些类进行配置，具体内容如代码 3.25 所示。

代码 3.25　封装类：applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
...
    <bean name="loginService"
        class="com.cjg.test.LoginService" />            <!-- 配置服务层 -->
    <!-- 配置 Action 对应 Bean，并注入 loginService -->
    <bean name="login" class="com.cjg.test.Login">
        <property name="service">
            <ref bean="loginService" />
        </property>
    </bean>
</beans>
```

（5）最后还需要在 struts.xml 文件中对名为 Login 的 Action 类进行配置，实现页面的跳转，具体内容如代码 3.26 所示。

代码 3.26　配置请求：struts.xml

```
<!DOCTYPE struts PUBLIC
       "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
       "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="default" extends="struts-default">
        <action name="login" class="login">   <!-- 配置关于 login 的 Action-->
            <result>/success.jsp</result>
            <result name="error">/error.jsp</result>
```

```
        </action>
    </package>
</struts>
```

当登录成功，则会转到名为 success.jsp 页面，该页面的具体内容如代码 3.27 所示；否则就会转到名为 error.jsp 页面，该页面的具体内容如代码 3.28 所示。

代码 3.27　正确页面：success.jsp

```
...
    <body>
        <center>验证正确。</center>
    </body>
...
```

代码 3.28　错误页面：error.jsp

```
...
    <body>
        <center>用户名或密码错误，请重新输入。</center>
    </body>
...
```

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/struts2spring/index.jsp，就会出现如图 3.27 所示的运行结果。填写相应的信息后单击“提交”按钮，如果信息正确，则会转入如图 3.28 所示的验证成功页面，否则就会转入如图 3.29 所示的失败页面。

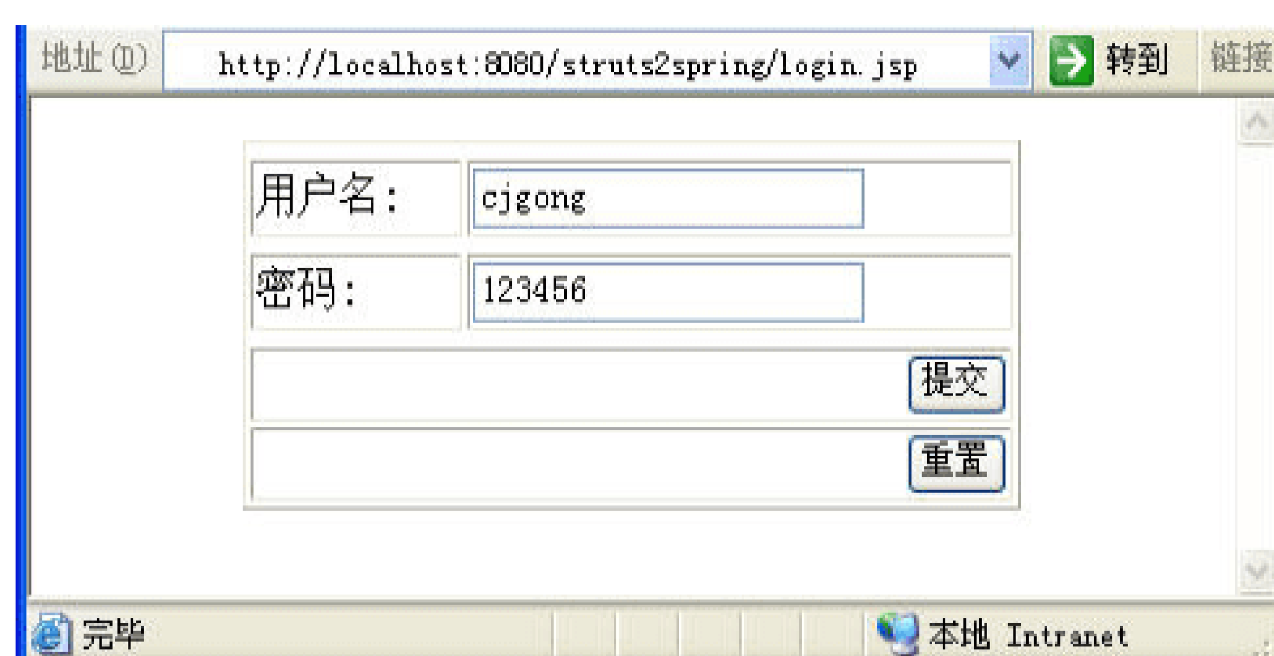


图 3.27　登录页面

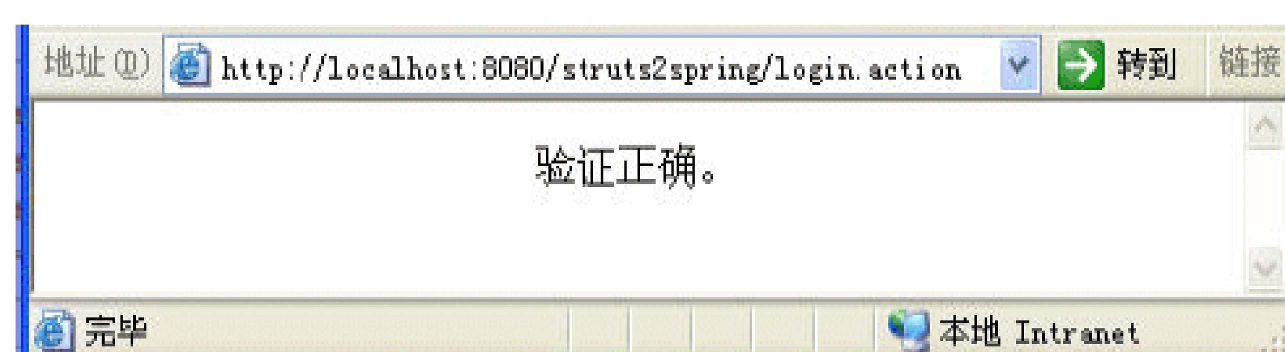


图 3.28　验证正确页面

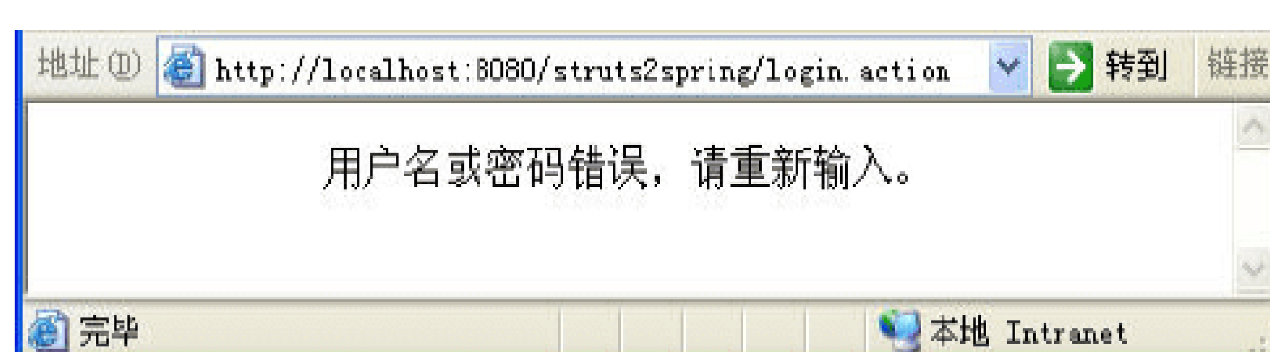


图 3.29　错误页面

3.4　实现 Spring、Struts 2.x 和 Hibernate 框架集成

通过上面的章节可以发现 Struts 1.x、Spring 和 Hibernate 3 个框架能够很容易实现集成，但是如果用 Struts 2.x 框架来代替 Struts 1.x，那么 3 个框架还能很好地集成吗？答案肯定是可以的，那么如何实现呢？本节将详细介绍该 3 个框架集成的具体实现过程。

3.4.1　Hibernate 与 Struts 2.x 框架集成

当 Hibernate 框架与 Struts 2.x 框架具体集成时，Struts 2.x 框架是不能直接与 Hibernate

框架发生联系的。之所以这样，因为表示层是不能与持久层发生联系，而 Struts 2.x 框架是作为表示层技术存在，Hibernate 框架则是作为持久层技术存在。为了解决该问题，必须要建立一个服务层，而 Struts 2.x 框架通过调用服务层间接调用持久层接口。

为了便于讲解，本节将利用 Struts 2.x 和 Hibernate 框架实现一个具体实例来介绍集成的详细过程。首先介绍代码的运行背景，即添加和显示数据库中的记录，具体步骤如下。

（1）在 MySQL 数据库中，创建数据库 name 和表格 name，具体内容如代码 3.29 所示。

代码 3.29　SQL 语句：name.sql

```
#创建 name 数据库
DROP DATABASE IF EXISTS 'name';
CREATE DATABASE 'name';
USE 'name';
# 创建 name 表，id 为主键
CREATE TABLE 'name' (
  'id' int(11) NOT NULL auto_increment,
  'name' varchar(50) default NULL,
  PRIMARY KEY  ('id')
) ENGINE=InnoDB DEFAULT CHARSET=gbk ROW_FORMAT=REDUNDANT;
```

（2）新建一个名为 struts2hibernate 的 Web Project 项目，接着引入 Struts 2.x 框架的相关 jar 文件，对于 Struts 2.x 框架需要如下：struts2-core-2.0.11.jar、xwork-2.0.4.jar、ognl-2.6.11.jar、freemarker-2.3.8.jar 和 commons-logging-1.0.4.jar。然后由于该框架需要与 Hibernate 框架集成，所以需要通过 MyEclipse 开发环境的向导添加关于 Hibernate 框架的支持，这时该项目的目录如图 3.30 所示。

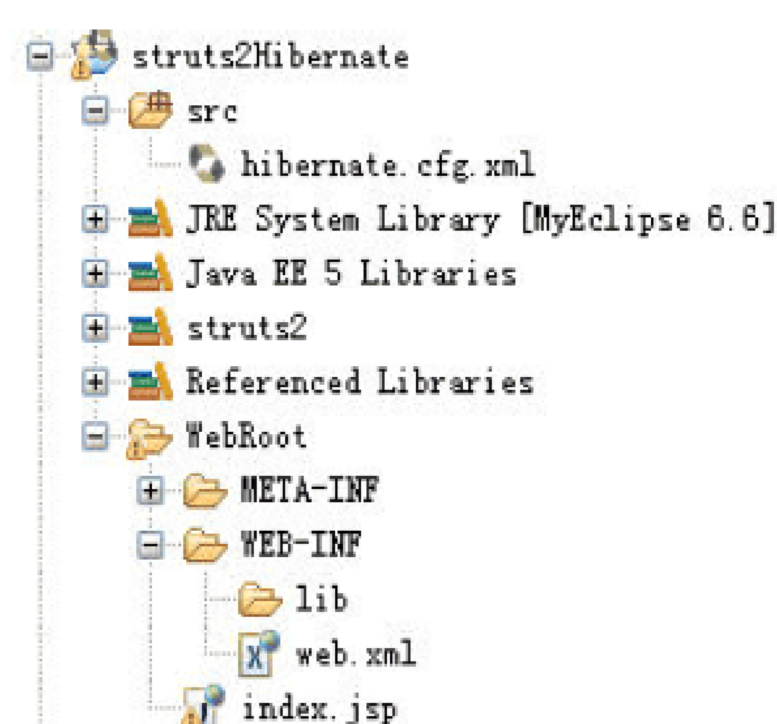


图 3.30　项目目录

（3）接着在 web.xml 中添加关于 Struts 2.x 的监听，然后通过 MyEclipse 开发环境的逆向工程实现关于数据库的持久层，关于数据库表格的实体类和映射文件分别如代码 3.30 和代码 3.31 所示。这时 hibernate.cfg.xml 配置文件的内容如代码 3.32 所示。

代码 3.30　实体类：Name.java

```
...
public class Name {
    //创建属性
    private Integer id;
    private String name;
    public Name() {                                      //构造函数
    }
    //配置相关属性的 get()和 set()方法
...
}
```

代码 3.31　实体类映射文件：Name.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping>
    <!--制定要持久化类与对应数据库的表映射关系-->
    <class name="com.cjg.domain.Name" table="name" catalog="name">
        <id name="id" type="java.lang.Integer">    <!--表 name 主键的设置>
            <column name="id" />
            <generator class="native" />
        </id>
        <!--表 name 字段 name 与属性 name 的映射关系>
        <property name="name" type="java.lang.String">
            <column name="name" length="50" />
        </property>
    </class>
</hibernate-mapping>
```

代码 3.32　Hibernate 框架配置文件：hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-
        3.0.dtd">
<hibernate-configuration>
<session-factory>
      <!-- 指定连接数据库用户名-->
      <property name="connection.username">root</property>
      <!-- 指定连接数据库 URL-->
      <property name="connection.url">jdbc:mysql://localhost:3306/name
      </property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect
      </property>
      <!-- 指定连接数据方言->
      <property name="myeclipse.connection.profile">MySQL</property>
      <!-- 指定连接数据库密码->
      <property name="connection.password">root</property>
      <!-- 指定连接数据库驱动->
      <property name="connection.driver_class">com.mysql.jdbc.Driver
      </property>
      <!-- 指定映射文件->
      <mapping resource="com/cjg/domain/Name.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

（4）创建服务层，在该服务层中通过调用 Hibernate 框架中的相关接口来实现数据的持久化操作。服务层由一个接口和实现该接口的类共同组成，具体内容分别如代码 3.33 和代码 3.34 所示。

代码 3.33　实现逻辑的接口类：NameService.java

```
public interface NameService {
    public void add(Name book);                 //插入信息操作
    public List find();                         //查询所有信息操作
}
```

代码 3.34　接口继续类：NameServiceHibernateImpl.java

```
...
public class NameServiceHibernateImpl implements NameService {
```

```
    //实例化 Book 对象的 session 工厂，以便调用 Hibernate 接口
    public static SessionFactory sessionFactory;
    static {
        try {
            Configuration config = new Configuration();
                                                    //创建 Configuration 对象
            config.addClass(Name.class);
            sessionFactory = config.buildSessionFactory();
                                                    //获取 SessionFactory 对象
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void add(Name name) {                    //实现增加注册记录
        Session session = sessionFactory.openSession();
        //声明并管理事务
        Transaction tx = null;
        try {
            tx = session.beginTransaction();        //开始事物
            session.saveOrUpdate(name);
            tx.commit();                            //提交事物
        } catch (Exception e) {
            if (tx != null) {
                tx.rollback();                      //回滚事物
            }
        } finally {
            session.close();                        //关闭 Session 对象
        }
    }
    public List find() {                            //遍历注册记录
        Session session = sessionFactory.openSession();//获取 Session 对象
        Transaction tx = null;                              //创建事物对象
        try {
            tx = session.beginTransaction();                //开始事务
            Query query = session.createQuery("from Name");
            List list = query.list();
            tx.commit();                            //提交事务
            return list;
        } catch (Exception e) {
            if (tx != null) {
                tx.rollback();                      //回滚事务
            }
        } finally {
            session.close();                        //关闭 Session 对象
        }
        return null;
    }
}
```

（5）创建实现页面跳转的 Action 类，在该类中不仅通过调用不同的方法实现各种功能，而且还实现了页面的跳转，该类的具体内容如代码 3.35 所示。

代码 3.35　实现页面跳转 Action：NameAction.java

```
public class NameAction extends ActionSupport {
    //创建字段
    private NameService service;
    private String name;
```

```
    private List list;
    public NameAction() {
        service = new NameServiceHibernateImpl();
    }
    public String addName() {                          //编写添加注册记录
        Name name1= new Name();
        name1.setName(name);
        service.add(name1);
        return SUCCESS;
    }
    public String ListName() {                         //编写遍历注册方法
        List l = service.find();
        setList(l);
        return SUCCESS;
    }
    public List getList() {                            //配置 list 属性
        return list;
    }
    public void setList(List list) {
        this.list = list;
    }
    public String getName() {                          //配置 name 属性
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

接着在 struts.xml 文件中配置该 Action 类，具体内容如代码 3.36 所示。

代码 3.36　Action 类的配置文件：struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
       "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
       "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!--设置该项目的编码方式 -->
    <constant name="struts.i18n.encoding" value="GBK" />
    <!--继承 struts 的配置文件 -->
    <package name="default" extends="struts-default">
        <!-- 配置关于 add 的 Action-->
        <action name="add" class="com.cjg.action.NameAction" method=
        "addName">
            <result type="chain">list</result>
        </action>
        <!-- 配置关于 list 的 Action-->
        <action name="list" class="com.cjg.action.NameAction" method=
        "ListName">
            <result>/success.jsp</result>
        </action>
    </package>
</struts>
```

【代码解析】

在上述代码中，当接收到名为 add 的请求时，则会用 com.cjg.action.NameAction 类中的 addName 方法来处理；当接收到名为 list 的请求时，则会用 com.cjg.action.NameAction

类中的 ListName 方法来处理。

（6）用户首先应该登录首页进行注册，该页面的具体内容如代码 3.37 所示。如果注册成功，则会转到显示所有注册信息的页面，该页面的具体内容如代码 3.38 所示。

代码 3.37　注册首面：index.jsp

```
...
    <body>
        <center>
            注册的名字：
            <s:form action="add">
                <s:textfield name="name" />
                <s:submit value="确定" />
            </s:form>
        </center>
    </body>
...
```

代码 3.38　注册成功页面：success.jsp

```
...
        <center>
            已注册的名字：
            <br>
            <table border="1">
                <s:iterator value="list">
                    <tr><td>
                            <s:property value="name" />
                    </td></tr>
                </s:iterator>
            </table>
        </center>
...
```

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/struts2Hibernate/index.jsp，运行结果如图 3.31 所示。填写相应的注册信息后，单击“确定”按钮，如果信息正确，则会转入图 3.32 所示的注册信息成功的页面。



图 3.31　首页



图 3.32　注册成功页面

3.4.2　Struts 2.x 和 JPA 框架集成

通过 3.4.1 节的讲解可以知道 Struts 2.x 框架能通过服务层实现与 Hibernate 框架的集成，那么当 JPA 框架代替 Hibernate 框架后，Struts 2.x 框架还可以与其实现集成吗？答案

肯定是可以，那么如何实现呢？本节将详细讲解该集成过程。

当 JPA 框架与 Struts 2.x 框架在具体集成时，Struts 2.x 框架中的某个方法可以直接调用 JPA 方法，具体步骤如下。

（1）首先在 SQL Server 2000 数据库中，创建数据库 testJpa 和表格 city，具体命令如代码 3.39 所示。

代码 3.39　SQL 语句：testJpa.sql

```
#创建 testJpa 数据库
DROP DATABASE IF EXISTS 'name';
CREATE DATABASE 'testJpa';
USE ' testJpa ';
# 创建 city 表，id 为主键
CREATE TABLE 'city' (
  'name' varchar (50) NULL,
  'id' int identity(1,1) Not NULL,
  PRIMARY KEY  ('id')
);
```

（2）新建一个名为 struts2jpa 的 Web Project 项目，接着引入 Struts 2 框架的相关 jar 文件，对于 Struts 2 框架需要如下：struts2-core-2.0.11.jar、xwork-2.0.4.jar、ognl-2.6.11.jar、freemarker-2.3.8.jar 和 commons-logging-1.0.4.jar。然后由于需要与 jpa 框架集成，所以需要通过 MyEclipse 开发环境的向导添加关于 jpa 框架的支持，这时该项目的目录如图 3.33 所示。

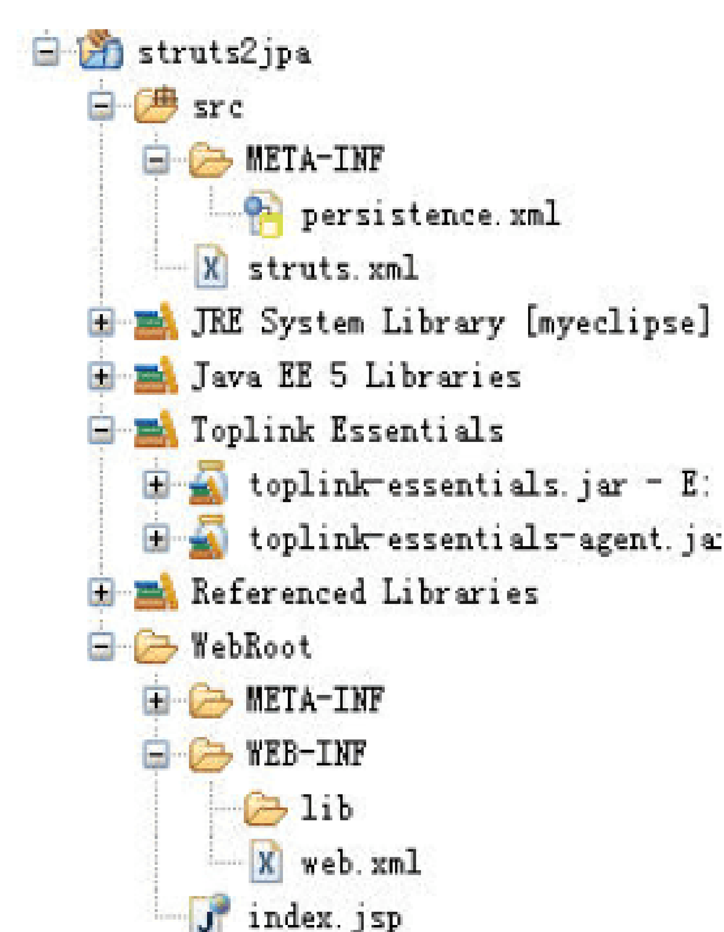


图 3.33　项目目录

（3）接着在 web.xml 中添加关于 Struts 2.x 的监听，然后再创建 struts.xml 文件。

（4）通过 MyEclipse 开发环境的逆向工程实现关于数据库的持久层，关于数据库表格的实体类如代码 3.40 所示，而关于项目的实体管理器的具体内容如代码 3.41 所示。

代码 3.40　持久层类：City.java

```
...
@Entity
@Table(name = "city")
public class City implements java.io.Serializable {
    //创建属性
    private Integer id;
    private String name;
    public City() {                                     //构造函数
    }
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "id")
    public Integer getId() {                            //配置属性 id
        return this.id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    @Column(name = "name")                              //配置属性 name
```

```
    public String getName() {
        return this.name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

代码 3.41　实体管理器：EntityManagerHelper.java

```
...
public class EntityManagerHelper {
    //创建属性
    private static final EntityManagerFactory emf;
    private static final ThreadLocal<EntityManager> threadLocal;
    static {                                            //静态模块
        emf = Persistence.createEntityManagerFactory("Struts2 JPAPU");
        threadLocal = new ThreadLocal<EntityManager>();
    }
    public static EntityManager getEntityManager() {    //获取实体管理器
        EntityManager manager = threadLocal.get();
        if (manager == null || !manager.isOpen()) {
            manager = emf.createEntityManager();
            threadLocal.set(manager);
        }
        return manager;
    }
    public static void closeEntityManager() {           //关闭实体管理器
        EntityManager em = threadLocal.get();
        threadLocal.set(null);
        if (em != null)
            em.close();
    }
    public static void beginTransaction() {             //开启事务
        getEntityManager().getTransaction().begin();
    }
    public static void commit() {                       //提交事务
        getEntityManager().getTransaction().commit();
    }
    public static void rollback() {                     //回滚事务
        getEntityManager().getTransaction().rollback();
    }
}
```

这时，关于 JPA 框架配置文件 persistence.xml 的具体内容如代码 3.42 所示。

代码 3.42　JPA 框架的配置文件：persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence 1 0.xsd"
    version="1.0">
    <persistence-unit name="Struts2 JPAPU"
        transaction-type="RESOURCE LOCAL">
        <provider>                              <!--配置管理器-->
            oracle.toplink.essentials.PersistenceProvider
```

```
        </provider>
        <class>com.cjg.persist.City</class>
        <properties>                    <!--配置数据库的相关信息-->
            <!--配置数据库驱动-->
            <property name="toplink.jdbc.driver"
                value="com.microsoft.jdbc.sqlserver.SQLServerDriver" />
            <!--配置数据库 URL-->
            <property name="toplink.jdbc.url"
                value="jdbc:microsoft:sqlserver://localhost:1433;
                DatabaseName=testJpa" />
            <!--配置数据库用户名-->
            <property name="toplink.jdbc.user" value="sa" />
            <!--配置数据库密码-->
            <property name="toplink.jdbc.password" value="root" />
        </properties>
    </persistence-unit>
</persistence>
```

（5）创建服务层，在该服务层中通过调用 JPA 框架中的相关接口来实现数据的持久化操作。具体内容如代码 3.43 所示。

代码 3.43　实现业务层：CityDAO.java

```
...
public class CityDAO {
    private EntityManager getEntityManager() {          //获取实体管理器
        return EntityManagerHelper.getEntityManager();
    }
    public void save(City entity) {                     //实现保存方法
        // 开始事务的方法
        EntityManagerHelper.beginTransaction();
        try {
            getEntityManager().persist(entity);         //在数据库中插入数据
            EntityManagerHelper.commit();               //事务提交的方法
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();             //事务回滚的方法
            throw re;
        }finally{
            EntityManagerHelper.closeEntityManager();
        }
    }
}
```

（6）创建实现页面跳转的 Action 类，在该类中不仅通过调用不同的方法实现各种功能，而且还实现了页面的跳转，该类的具体内容如代码 3.44 所示。

代码 3.44　实现页面跳转的 Action 类：CityAction.java

```
...
public class CityAction {
    City city;                                  //创建属性

    public City getCity() {                     //配置属性 city
        return city;
    }
    public void setCity(City city) {
```

```
        this.city = city;
    }
    public String execute(){                          //编写执行方法
        CityDAO cd=new CityDAO();
        cd.save(city);
        return Action.SUCCESS;
    }
}
```

接着在 struts.xml 文件中配置该 Action 类，具体内容如代码 3.45 所示。

代码 3.45　实现对请求的配置：struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="jpa" extends="struts-default">
        <action name="insert" class="com.cjg.action.CityAction">
                                                <!-- 配置 Action-->
            <result>/success.jsp</result>
        </action>
    </package>
</struts>
```

从上述代码中可以发现，当用户登录首页后就会发出请求，该页面的具体内容如代码 3.46 所示。当登录成功后，则会转到名为 success.jsp 页面，该页面的具体内容如代码 3.47 所示。

代码 3.46　发生请求页面：index.jsp

```
...
    <body>
    <s:form action="insert">              <!--表单的处理类-->
        <s:textfield name="city.name" label="输入所在城市"></s:textfield>
        <s:submit value="提交"></s:submit>
    </s:form>
    </body>
...
```

代码 3.47　成功页面：success.jsp

```
...
    <body>
    信息成功
    </body>
...
```

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8081/struts2jpa/index.jsp，运行结果如图 3.34 所示。填写相应的信息后单击“提交”按钮，如果信息正确时，则会转入如图 3.35 所示的添加成功页面。

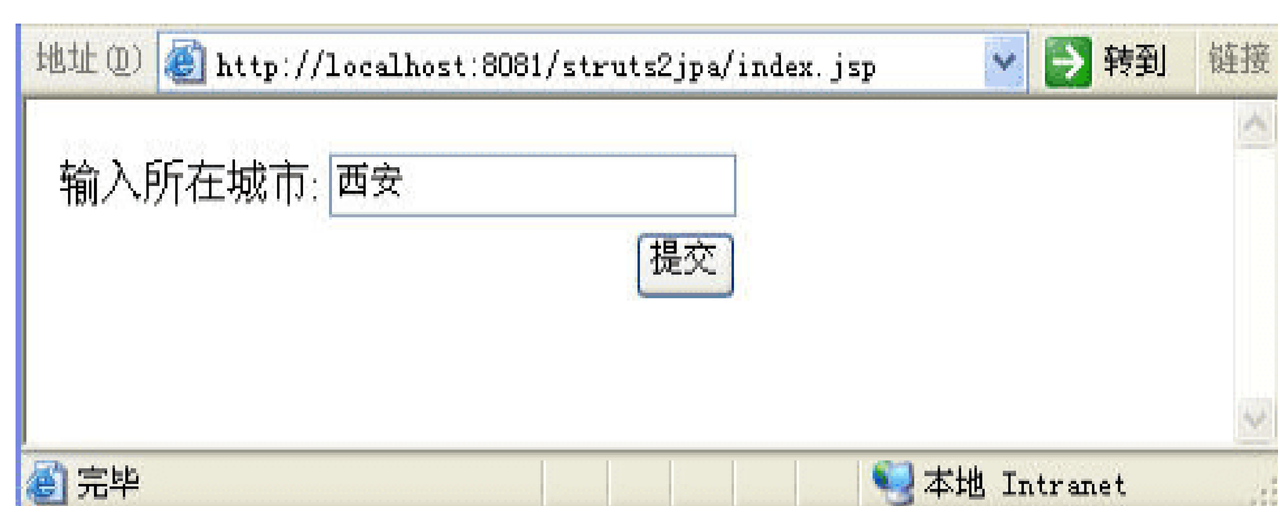


图 3.34　首页

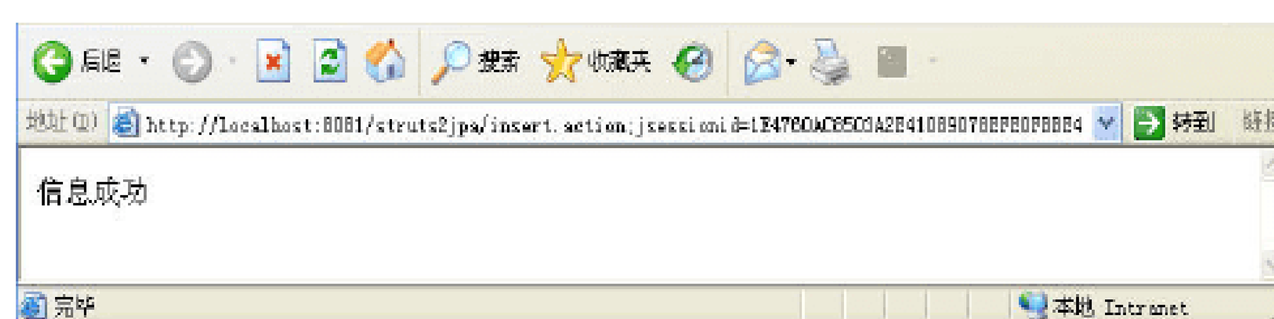


图 3.35　添加成功

3.4.3　Struts 2.x、Spring 和 Hibernate 框架集成

由于 Struts 2.x+Spring 和 Struts 2.x+Hibernate 集成时都是通过一个具体的实例来讲解，所以本节将不通过具体的实例来讲解如何实现 Struts 2.x+Spring+Hibernate 框架的集成，而是只讲解一下三者集成环境的实现。具体步骤如下：

（1）新建一个名为 strutsspringhibernate2 的 Web Project。

（2）为项目增加 Hibernate 框架相关类库与文件，完成支持 Hibernate 框架开发的环境后，其目录结构如图 3.36 所示。

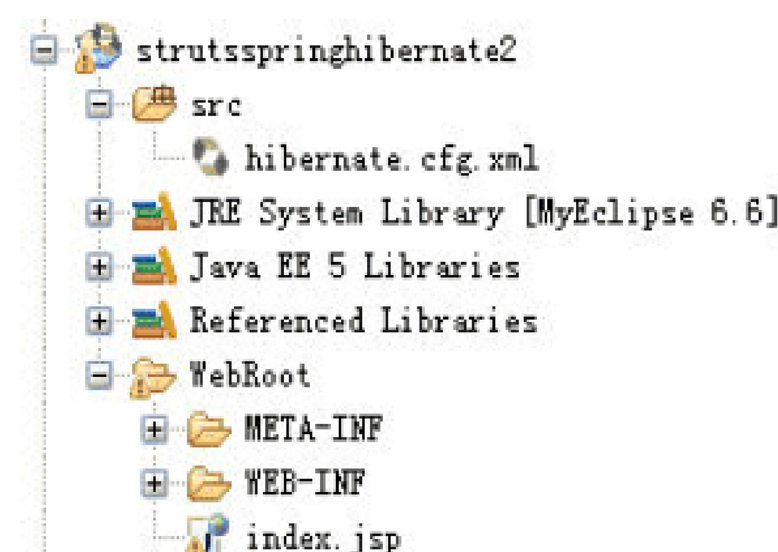


图 3.36　目录结构

（3）为项目增加 Spring 框架相关类库与文件，在具体配置时只需在第 3 页创建 SessionFactory 类时，取消 Create Spring SessionFactory that references 复选框的选择，如图 3.37 所示。完成支持 Spring 框架和 Hibernate 框架集成开发的环境后，其目录结构如图 3.38 所示。

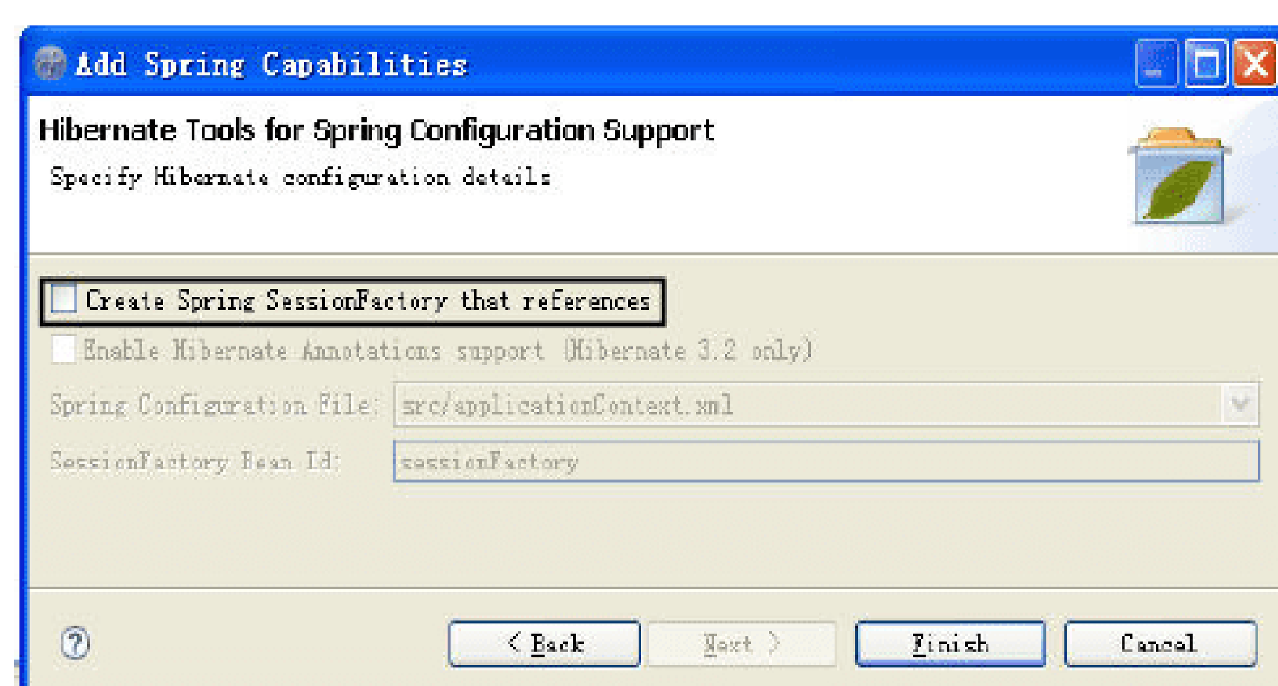


图 3.37　SessionFactory 类

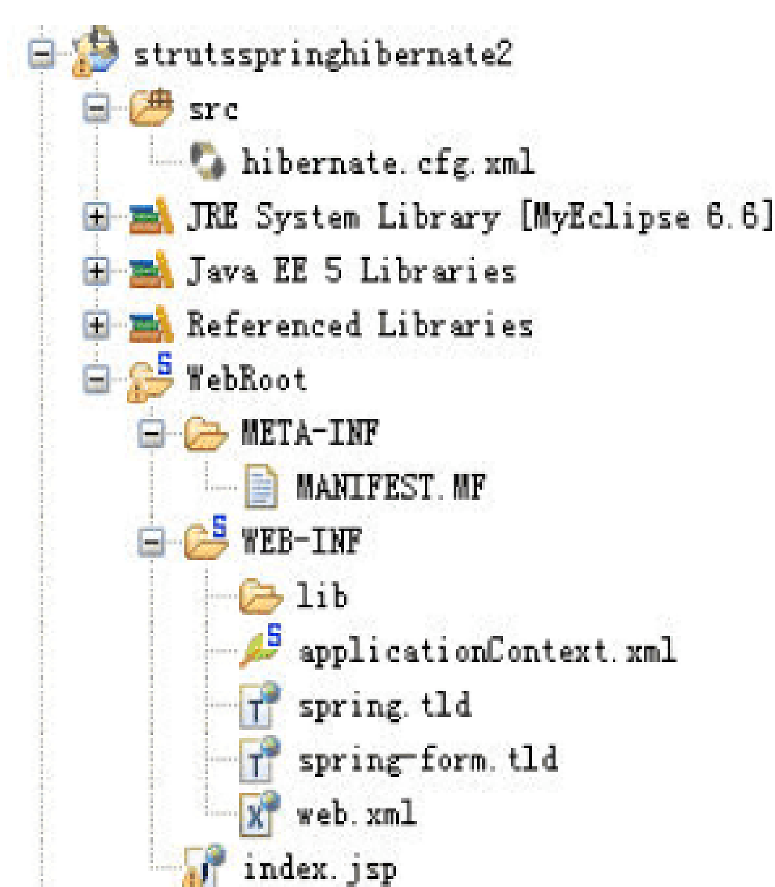


图 3.38　目录结构

（4）为项目增加 Struts 2.x 框架相关类库与文件，使 strutsspringhibernate2 项目支持 Struts 2.x 框架、Spring 框架与 Hibernate 框架三者集成。先把 Struts 2.x 框架的核心类库即将 Struts 2.1.6 框架下 lib 路径下的 struts2-core-2.1.6.jar、xwork-2.1.2.jar、ognl-2.0.8.jar、freemarker-2.3.13.jar 和 commons-logging-1.0.4.jar 添加到 Package Explorer 视图的 strutsspringhibernate2/WebRoot/WEB-INF/lib 的目录下，除了这 5 个必要包外还要增加 struts2-spring-plugin-2.0.11.jar 这个包，该包用来实现 Struts 2.x 框架与 Spring 框架的集成。

（5）接着添加 struts.xml 文件和修改 web.xml 文件，使的该项目支持 Struts 2.x 框架。首先修改 web.xml 文件，具体内容如代码 3.48 所示。

代码 3.48　配置 web.xml 文件：web.xml

```
<web-app >
    <context-param>                    <!--配置上下文对象-->
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/applicationContext.xml,classpath:applicationContext.
            xml
        </param-value>
    </context-param>
    <listener>                         <!--配置监听上下文对象-->
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    <filter>                           <!--配置 lazyLoadingFilter 过滤器-->
        <filter-name>lazyLoadingFilter</filter-name>
        <filter-class>
            org.springframework.orm.hibernate3.support.OpenSessionIn-
            ViewFilter
        </filter-class>
    </filter>
    <filter>                           <!--配置 struts2 过滤器-->
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <filter-mapping>                   <!--配置 lazyLoadingFilter 映射路径-->
        <filter-name>lazyLoadingFilter</filter-name>
        <url-pattern>*.action</url-pattern>
    </filter-mapping>
    <filter-mapping>                   <!--配置 struts2 映射路径-->
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
...
</web-app>
```

【代码解析】

上下文元素<context-param>用来设置 Spring 框架配置文件的路径，使项目启动时能够找到该配置文件。紧接着该元素要设置一个监听元素<listener>，使项目启动时能够加载上面的上下文对象。最后再增加一个关于 OpenSessionInViewFilter 的过滤器和该过滤器的映射。

注意：关于 OpenSessionInViewFilter 的过滤器不仅要放在 struts2 过滤器的前面，而且关于其的过滤器映射也要放在 struts2 过滤器映射的前面。

（6）最后，在 strutsspringhibernate2/src 目录下创建 struts.xml 文件，具体内容如代码 3.49 所示。

代码 3.49　配置 struts.xml 文件：struts.xmll

```
<struts>
    <constant name="struts.objectFactory" value="spring" />
    <package name="com_cjg" extends="struts-default"
```

```
...
    </package>
</struts>
```

【代码解析】

元素<constant>用来定义常量，其属性 struts.objectFactory 的值，用来表示当前 Struts 2.x 框架中的 Action 由 Spring 来管理，该元素要放在元素<package>的前面。

完成支持 Struts 2.x 框架、Spring 框架和 Hibernate 框架集成的开发环境后，其目录结构如图 3.39 所示。至此，该项目已经具备 Struts 2.x 框架、Spring 框架和 Hibernate 框架的技术支持。

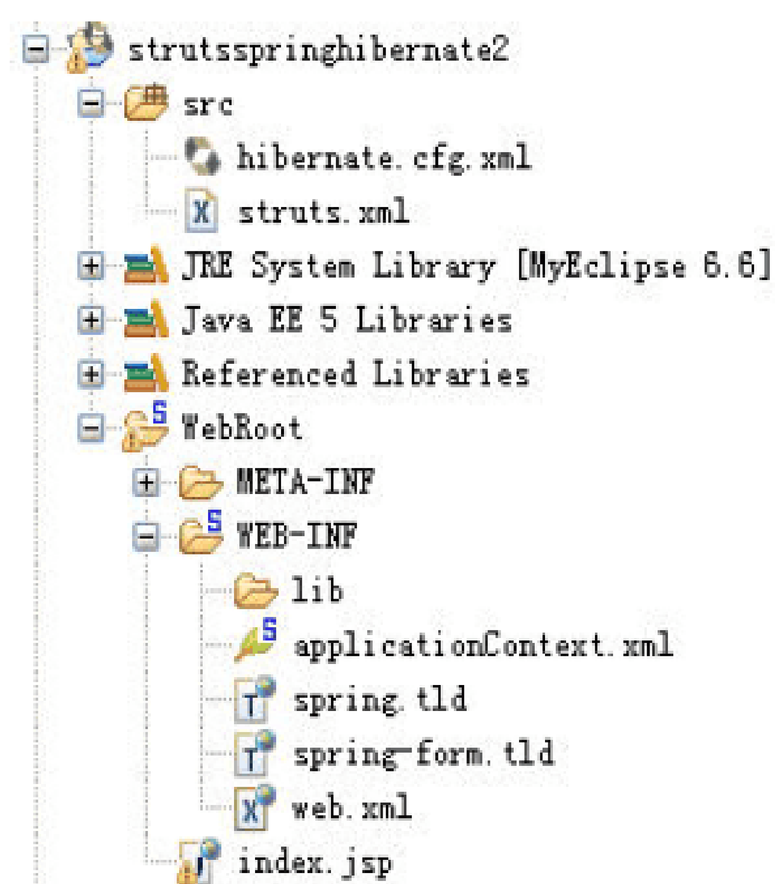


图 3.39　目录结构

3.5　小　　结

本章主要介绍在 MyEclipse 开发环境中实现各种框架的集成，这些框架主要是 Spring、Struts 和 Hibernate 框架。其中 Struts 框架不仅可以是 Struts 1.x 框架，而且还可以是 Struts 2.x。由于本书主要使用 Struts 2.x 框架，所以将把重心放在 Struts 2.x 上。

除了介绍 SSH 方面的各种集成外，本章还详细介绍了如何实现 JPA 框架与 Struts 2.x 框架的集成。

第 2 篇　典型模块开发

第 4 章　在线文本编辑器（FCKeditor）

在线文本编辑器对于任何网络系统来说是一个常见而不可缺少的模块，对于网络系统的表单页面程序来说几乎是必需模块。到目前为止，市场上比较流行的在线文本编辑器有：FCKeditor 在线文本编辑器、eWebEditor 在线文本编辑器和 tinyMCE 在线文本编辑器。

本章将详细介绍 FCKeditor 在线文本编辑器的各个方面：如何在表单页面中调用 FCKeditor 在线文本编辑器、如何配置出适合用户的在线文本编辑器、如何实现文件的上传功能和如何配置所要上传文件的类型等。

4.1　分析 FCKeditor 在线文本编辑器

FCKeditor 在线文本编辑器是一个比较成熟的开源产品，其不仅是一个非常强大的在线文本编辑器，而且还支持多种浏览器如：IE 5.5+、Firefox 1.5+、Safari 3.0+、Opera 9.50+、Netscape 7.1+和 Camino 1.0+等。

4.1.1　FCKeditor 在线文本编辑器功能描述

本节将以直观的方式向读者介绍 FCKeditor 在线文本编辑器可以实现的功能。这些功能包括设置输入文字格式、插入表情图标和上传文件、图像等。当浏览者浏览带有 FCKeditor 在线文本编辑器的页面时，显示页面如图 4.1 所示。

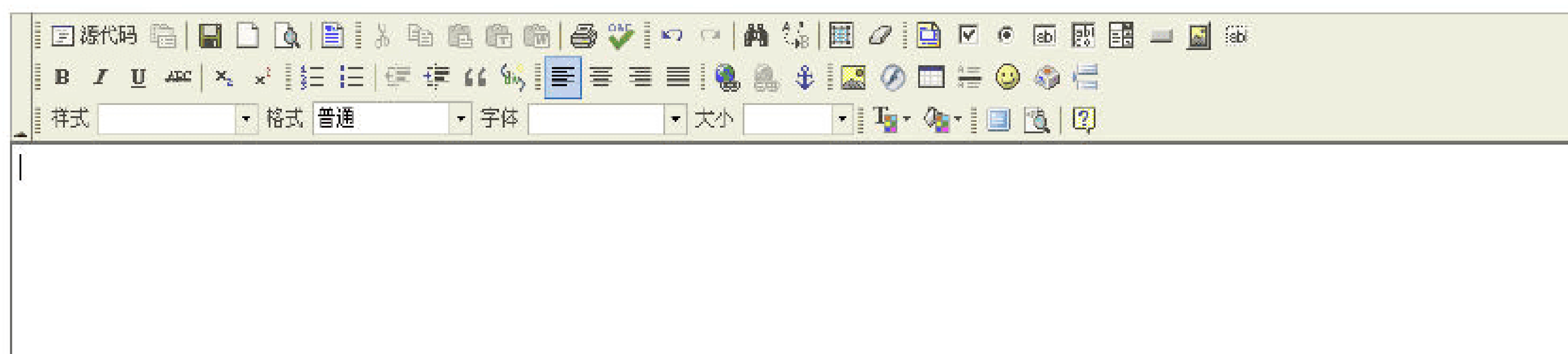


图 4.1　FCKeditor 在线文本编辑器

（1）在上述文本编辑器页面中，可以使用该编辑器对填写的内容进行格式的设置，如图 4.2 所示。

（2）在上述文本编辑器页面中，如果想插入表情图标，单击按钮就会弹出如图 4.3 所示的对话框。然后在该对话框中随便单击一个表情图标就会出现在文本编辑器中。

（3）在上述文本编辑器页面中，如果想插入图像，需要两个步骤：首先需要把图像上传到服务器上，然后在文本编辑器中选择上传的文件。例如插入一张名为 gougou 的图像，具体步骤如下：

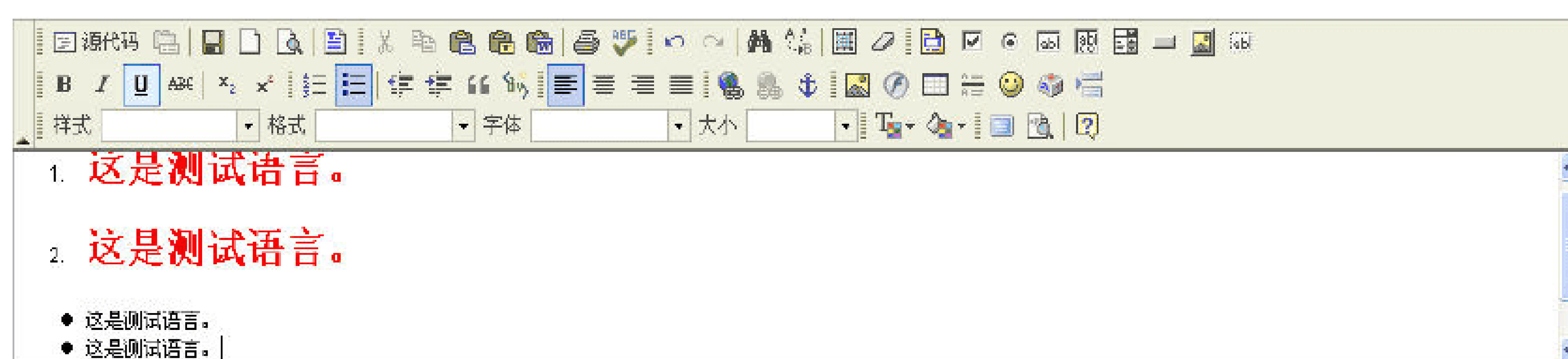


图 4.2　对文本进行格式设置

① 上传到服务器。首先单击按钮弹出如图 4.4 所示的插入图像对话框。在该对话框中单击“浏览服务器”按钮，弹出上传文件对话框（如图 4.5 所示）。在该对话框中只需单击“浏览”按钮选择相应的图像，然后单击 Upload 按钮就会上传所选的图像。





图 4.3　插入表情图标　　　　图 4.4　插入图像对话框

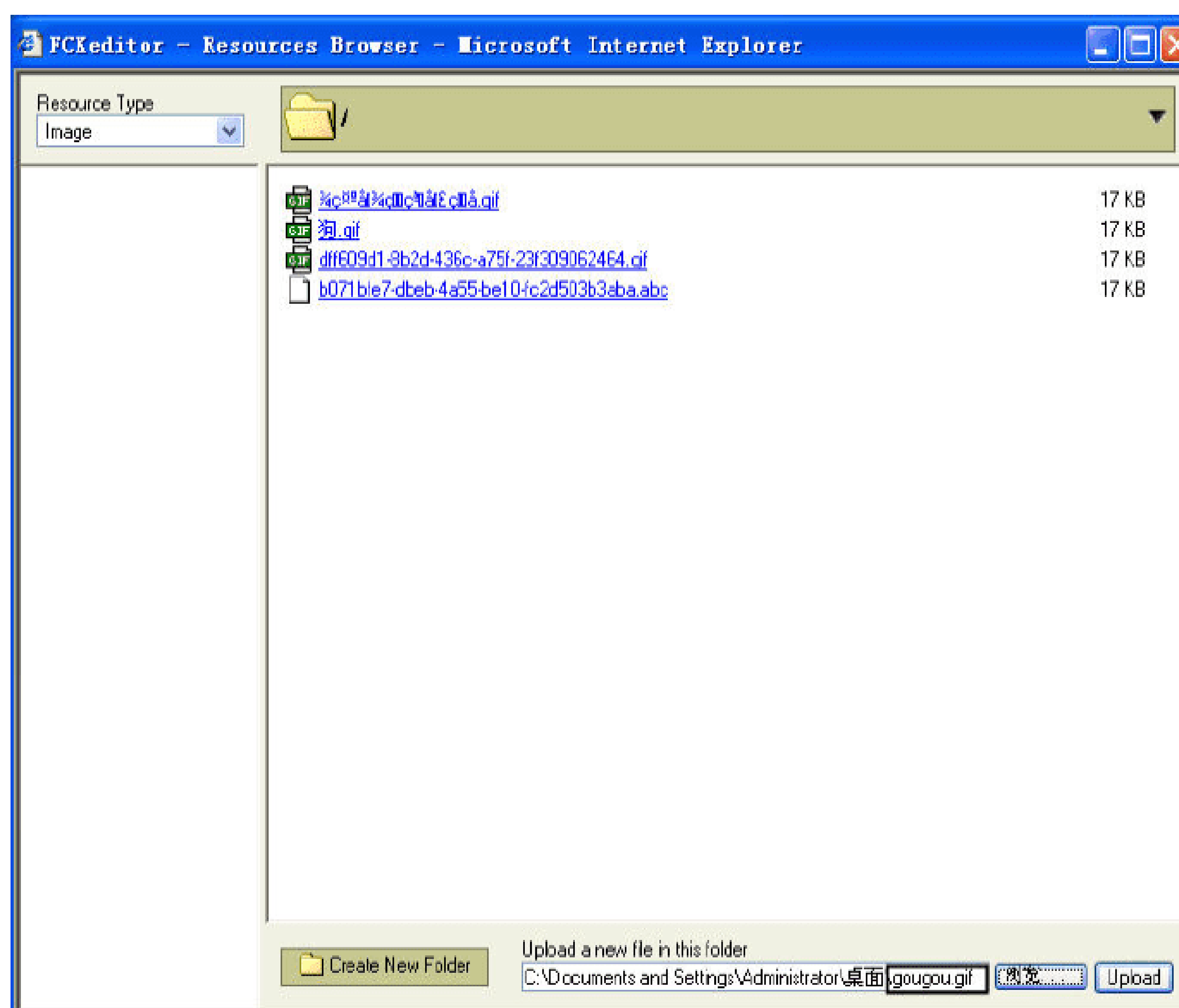


图 4.5　上传文件对话框

② 选择上传文件。在“上传文件”对话框中选择 gougou.gif 链接，就会在图 4.6 的对话框中显示出选择好的图像，最后单击“确定”按钮就会在在线文本编辑器中显示出正确

的图像（如图 4.7 所示）。

注意：如果想插入 Flash 影片，只需要单击按钮而其他步骤跟插入图像步骤一样。



图 4.6　选择相应图像

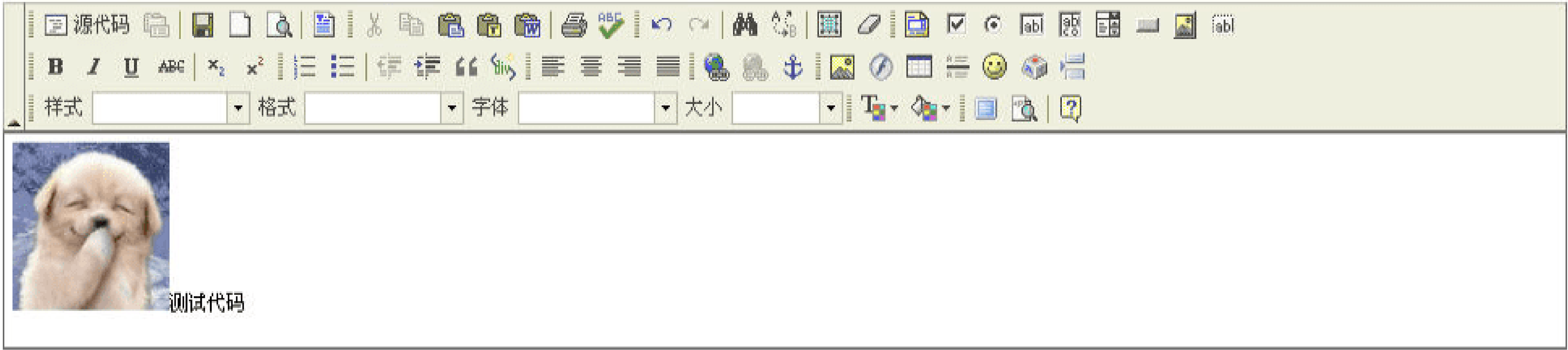


图 4.7　显示正确图像

（4）通过单击上传文件对话框的 Resource Type 下拉列表框，就会出现如图 4.8 所示的选项。通过它可以知道 FCKeditor 在线文本编辑器允许上传的类型有文件（File）、图像（Image）、动画（Flash）和视频（Media）。

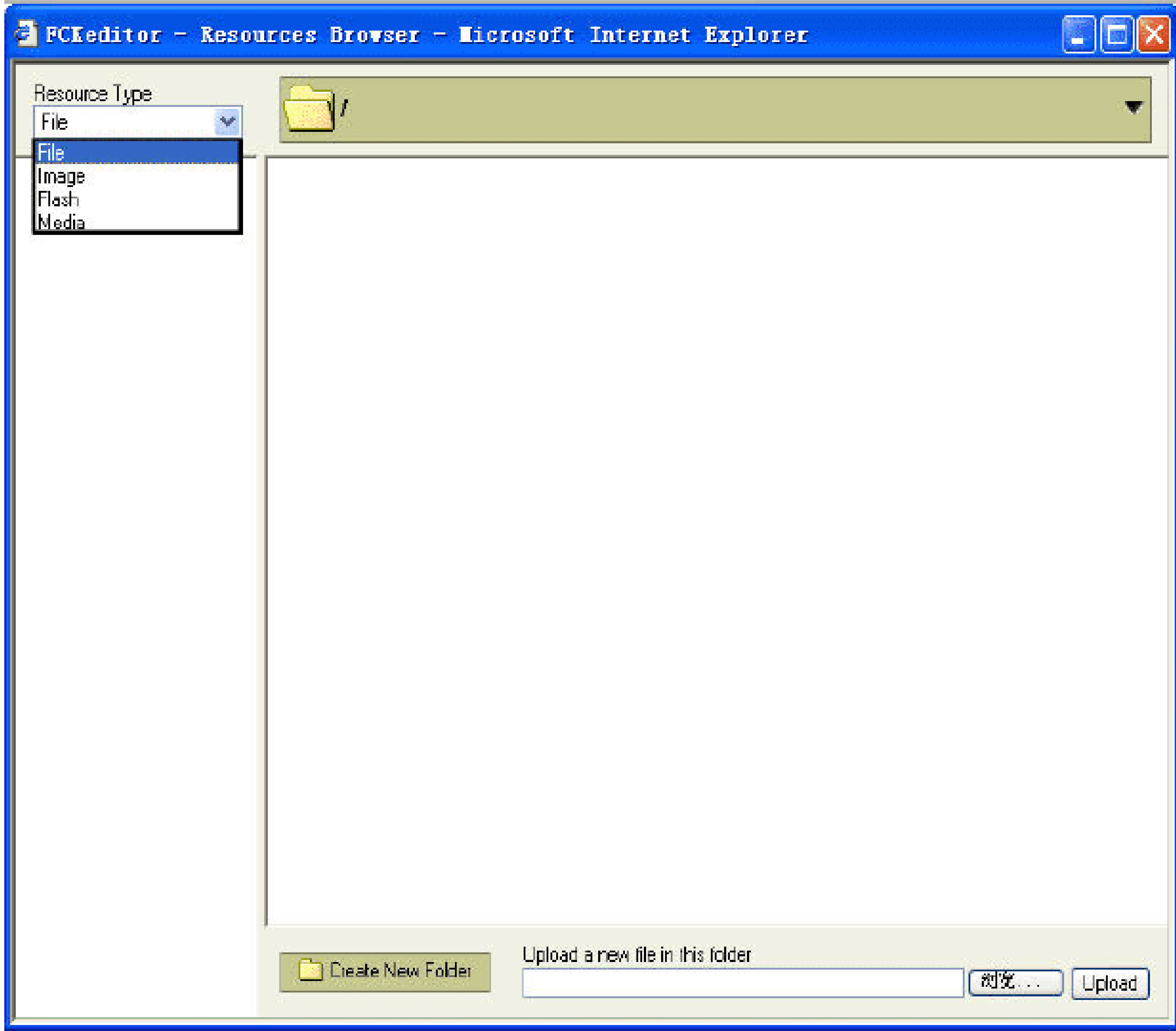


图 4.8　允许上传的类型

4.1.2　下载 FCKeditor 在线文本编辑器相关软件

FCKeditor 全称是在线文本编辑器，其名称中的 FCK 字符串是该编辑器作者名字 Frederico Caldeira Knabben 的缩写。目前最新的版本为 2.6.4 版本，具体的下载步骤如下。

（1）首先访问下载 FCKeditor 在线文本编辑器的官方网站（http://www.fckeditor.net/），如图 4.9 所示。

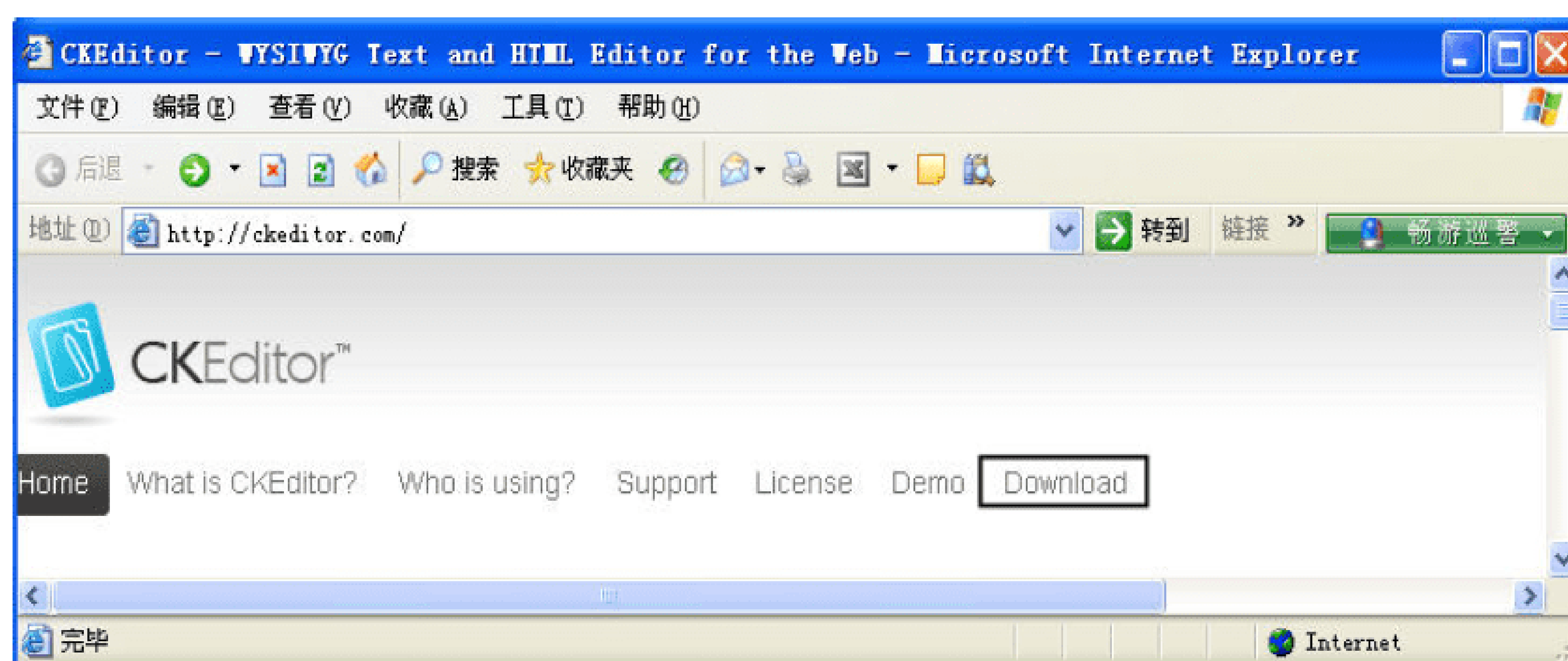


图 4.9　FCKeditor 编辑器首页

（2）打开 FCKeditor 在线文本编辑器官方网站的首页后，在其右上边的导航栏中选择 Download 选项，就会进入 FCKeditor 在线文本编辑器下载页面，如图 4.10 所示。

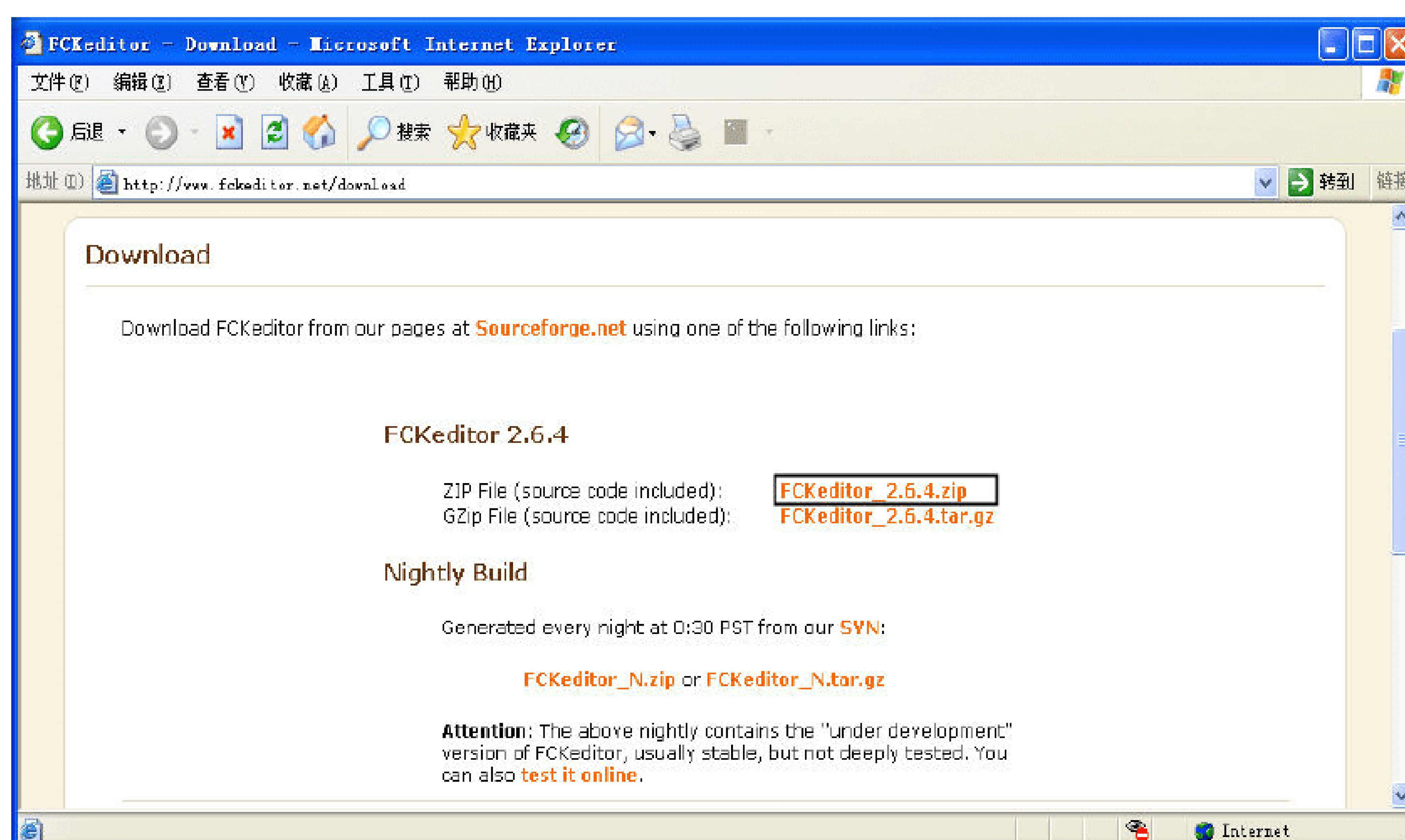


图 4.10　FCKeditor 编辑器下载页面

（3）在 FCKeditor 在线文本编辑器下载页面中，单击 FCKeditor_2.6.4.zip 链接后，就可以进入 FCKeditor 在线文本编辑器的真正下载页面（如图 4.11 所示），在该页面中单击 direct link 链接后，就可以下载该编辑器。

（4）为了使 FCKeditor 在线文本编辑器能够与服务器端进行交互，还需要下载 FCKeditor.java、该 jar 文件的源代码和 fckeditor-java-demo-2.5war 测试项目。FCKeditor.java 的最新版本为 2.5，在图 4.12 中的下面单击 Click here to download the latest version 链接后，

就可以进入下载 FCKeditor.java 相关 jar 文件的页面。

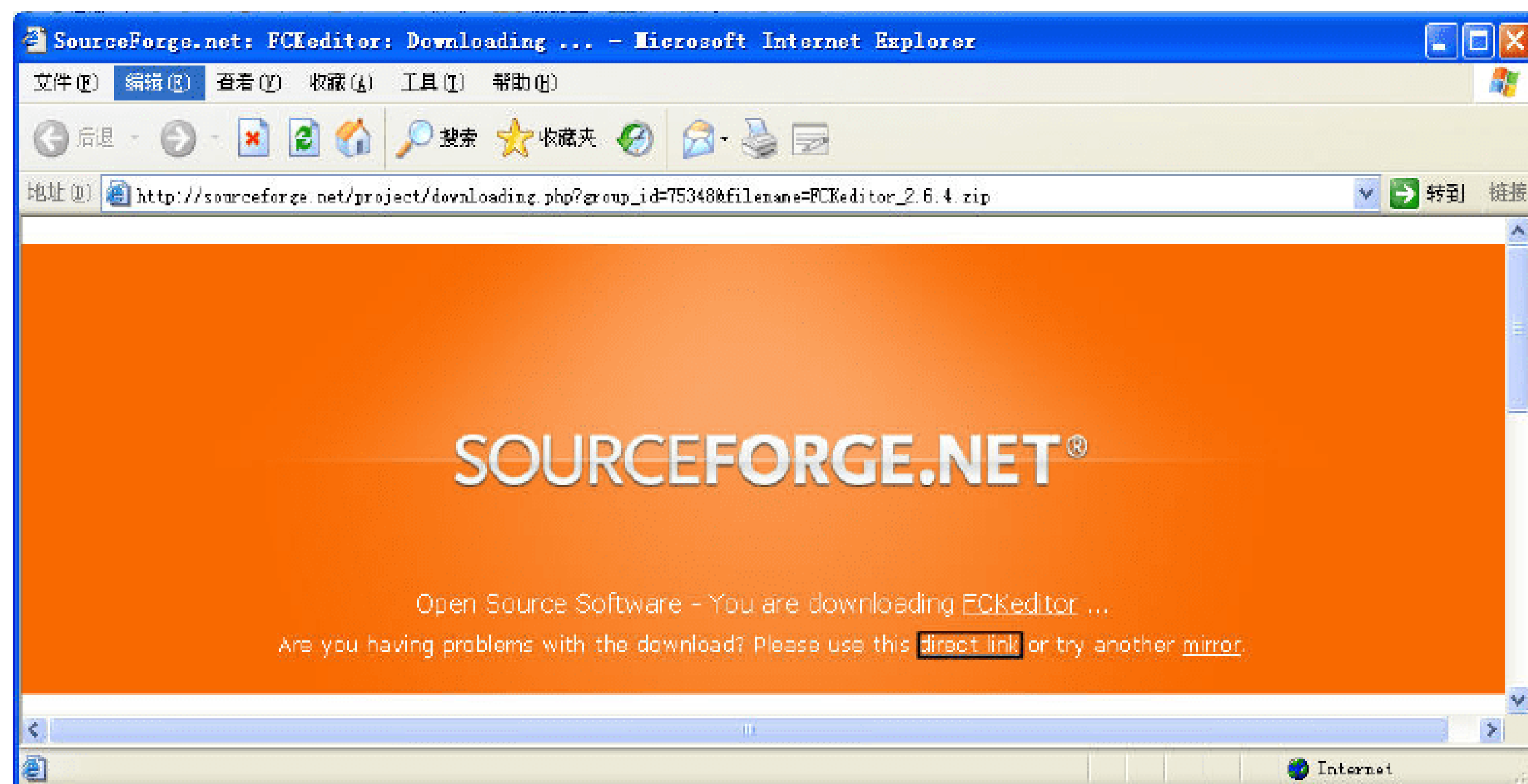


图 4.11　FCKeditor 编辑器真正下载页面

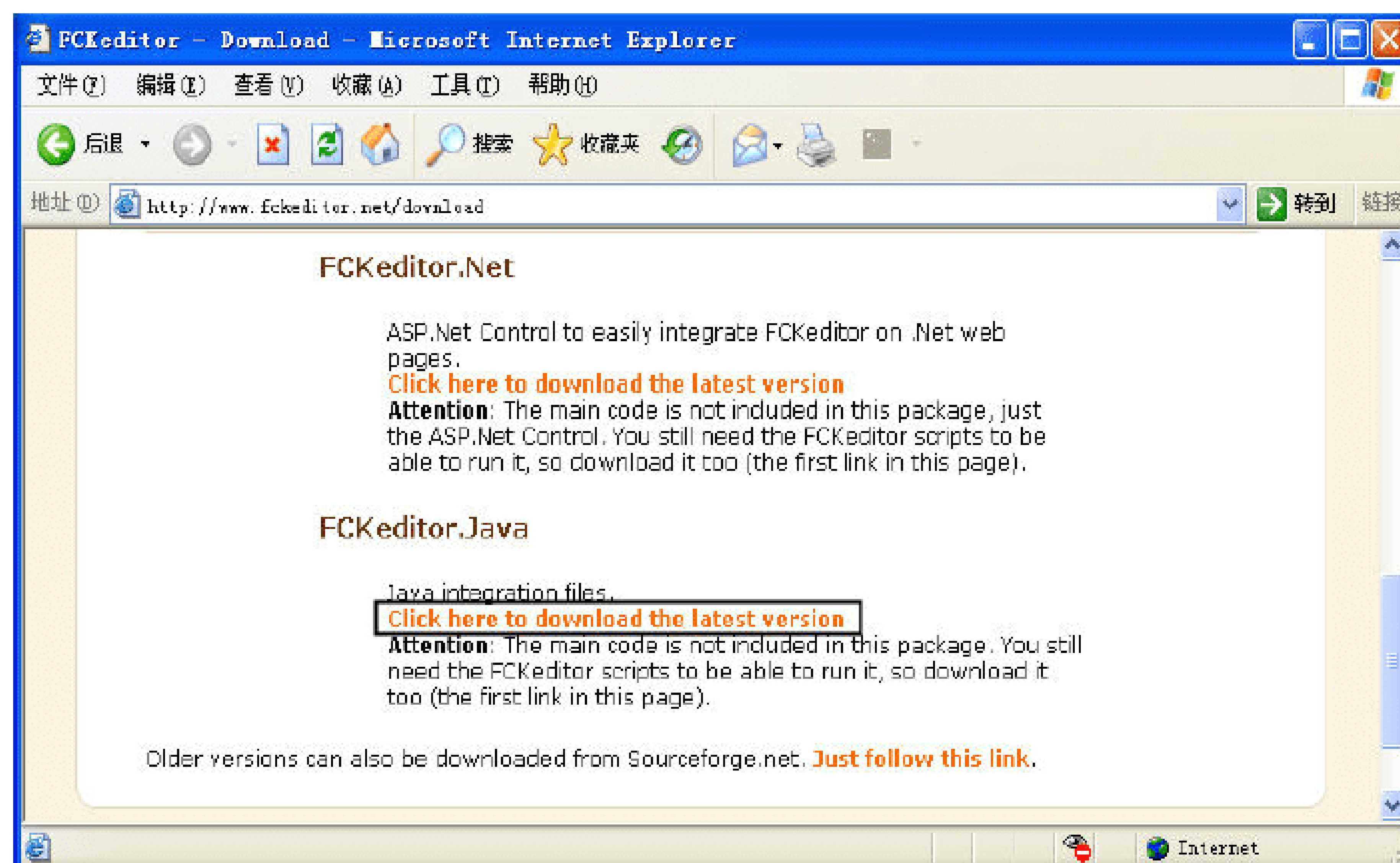


图 4.12　FCKeditor.java 页面

（5）接着在出现的下载页面（如图 4.13 所示）中，选择 fckeditor-java-2.5-bin.zip、fckeditor-java-2.5-src.zip 和 fckeditor-java-demo-2.5.war 链接后，就会进入各自真正下载的页面。在这些页面中单击 direct link 链接后，就可以下载这些 jar 文件。

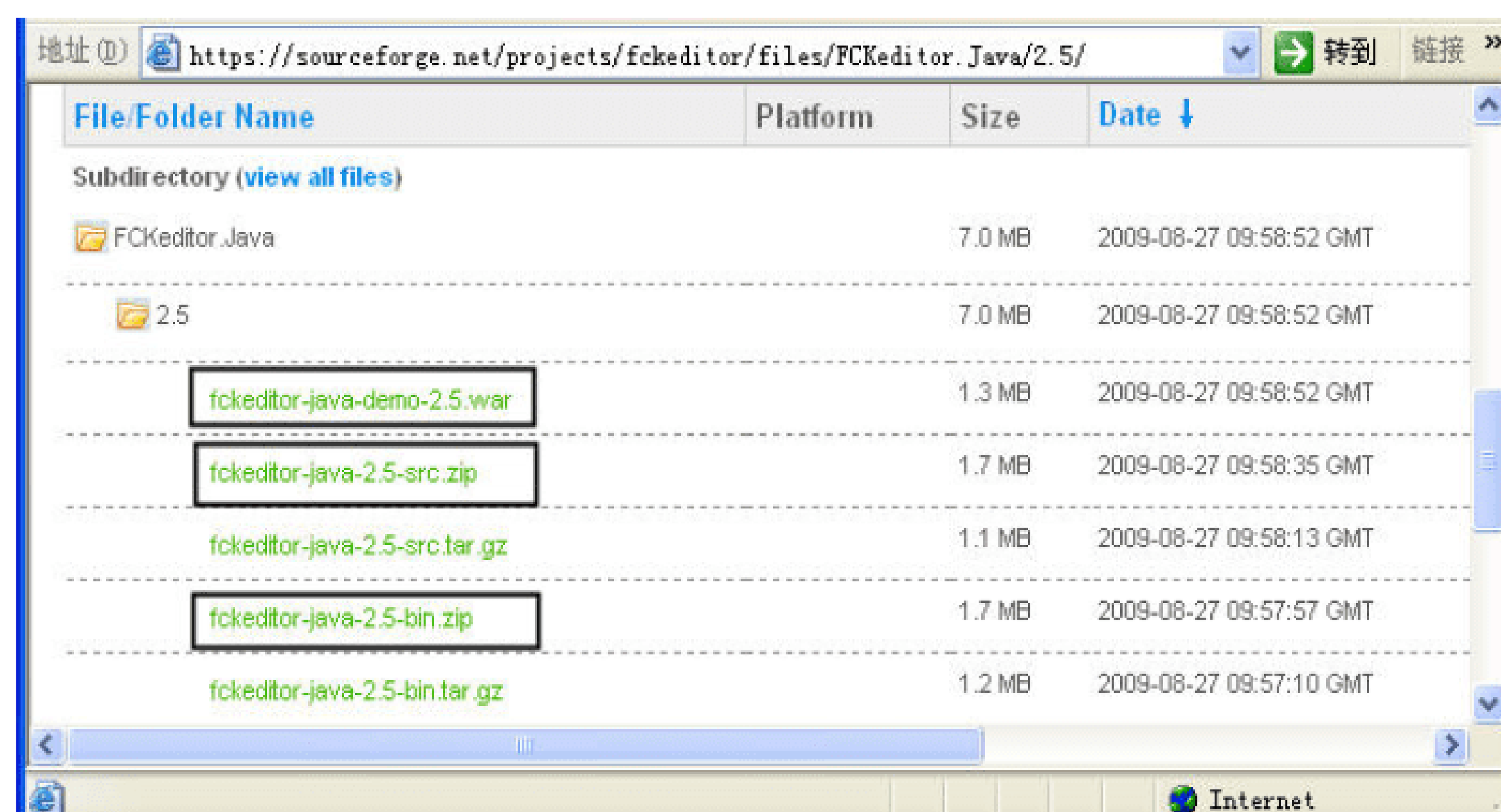


图 4.13　选择相应软件

至此，就完成对 FCKeditor 在线文件编辑器相关 jar 文件的下载。其中 FCKeditor_2.6.4.zip 文件为 FCKeditor 在线文件编辑器、fckeditor-java-2.5-bin.zip 文件为连接服务器的 jar 文件、fckeditor-java-2.5-src.zip 文件是 fckeditor-java-2.5-bin.zip 的源代码和 fckeditor-java-demo-2.5.war 文件为连接服务器的测试项目。

4.1.3　FCKeditor 在线文本编辑器目录简介和开发文档

4.1.2 节介绍了如何下载 FCKeditor 在线文本编辑器，下载完该框架后就可以在 Java Web 项目中使用该编辑器。在具体使用之前，先解压 FCKeditor_2.6.4.zip 文件，该压缩包目录如图 4.14 所示。

名称	大小
..	
editor	
_samples	
license.txt	67,713
fckutils.cfm	2,400
fcktemplates.xml	2,959
fckstyles.xml	3,450
fckpackager.xml	13,201
fckeditor_php5.php	6,156
fckeditor_php4.php	6,182
fckeditor.py	4,371
fckeditor.pl	3,506
fckeditor.php	994
fckeditor.lasso	3,949
fckeditor.js	9,606
fckeditor.cfm	7,024
fckeditor.cfc	8,830
fckeditor.asp	6,508
fckeditor.afp	4,307
fckconfig.js	13,927
_whatsnew_history.html	300,690
_whatsnew.html	9,837
_upgrade.html	1,359
_documentation.html	1,262

图 4.14　目录结构

在上述目录中由两个文件构成了 JSValidation 框架的主体结构，其他都是 Demo（用作演示）。这两个文件分别如下。

- _samples 文件：该文件下是关于 FCKeditor 在线文本编辑器的示例。
- editor 文件：该文件下是实现 FCKeditor 在线文本编辑器功能的主要文件。

此外，从该目录其他文件的后缀名中不难看出，FCKeditor 在线文本编辑器提供支持 asp、php、per 和 python 等各种服务器技术的版本，但是却不支持.NET 和 Java Web。如果想实现 FCKeditor 与 Java Web 之间的整合，则必须引入 jar 文件 P——fckeditor-java.zip。解压 fckeditor-java-2.5-bin.zip 文件，该压缩包目录如图 4.15 所示。

对 FCKeditor 在线文本编辑器进行精简，不仅可以提高 FCKeditor 在线文本编辑器的加载速度，而且还可以提高 Java Web 项目的运行速度。对其精简的步骤如下：

（1）对于图 4.14 的目录可以先删除_samples 文件，然后把后缀名为 js 和 xml 的所有文件全部删除，最后目录如图 4.16 所示。

（2）接着打开 editor 文件夹，在该文件中首先删除用来存储插件的 skins 文件夹和用来存储源代码的_source 文件夹，然后打开 lang 文件夹，因为一般只需用到中文和英文，所以除 en.js、zh.js 和 zh-cn.js 这 3 个文件外其他文件全部删除。最后在用来存储皮肤的 skins 文件中，删除用来存储相应皮肤的 silver 文件和 Microsoft Office 2003 文件。

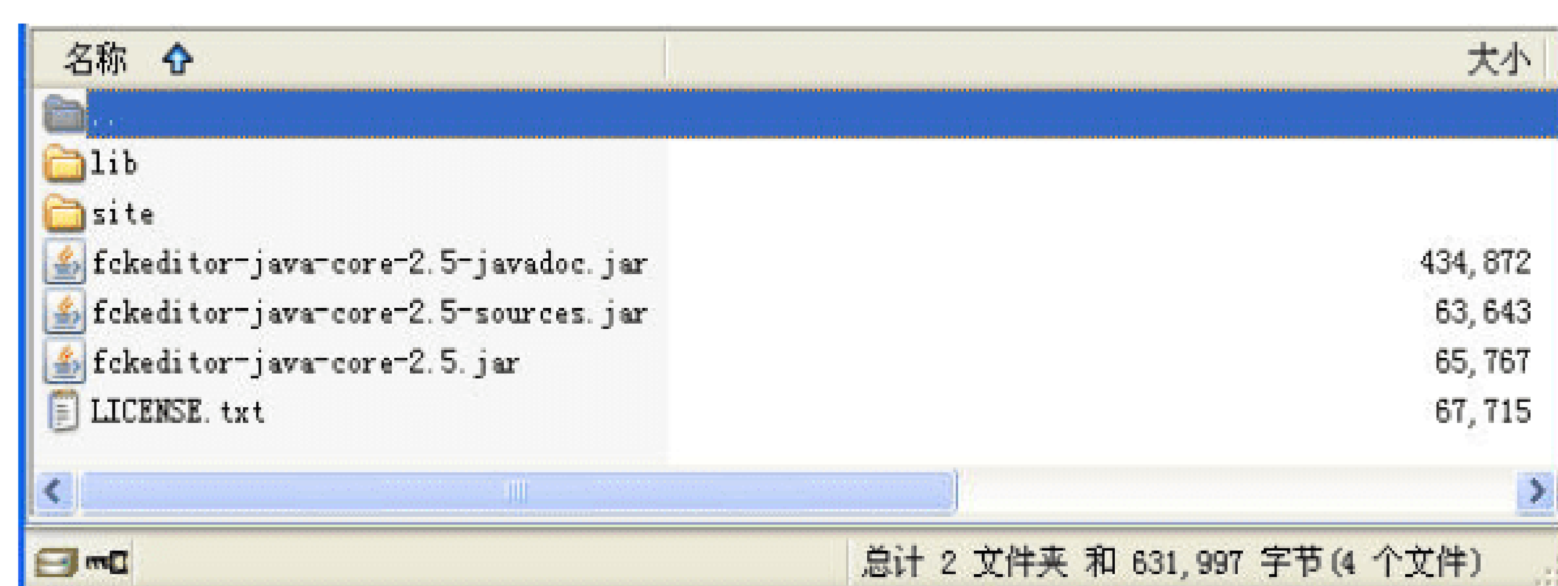


图 4.15　目录结构

名称	大小
editor	
fckconfig.js	14 KB
fckeditor.js	10 KB
fckpackager.xml	13 KB
fckstyles.xml	4 KB
fcktemplates.xml	3 KB

图 4.16　精简后目录

对于一个不熟悉的软件，如果想使用它，可以通过查看软件自带的开发文档和运行 demo 来学习。

对于开发不连接服务器的 FCKeditor 在线文本编辑器，可以通过访问该编辑器的开发文档地址（http://docs.fckeditor.net/）来打开开发文档页面，如图 4.17 所示。在该图中单击 Developer’s Guide 链接就可以进入开发文档目录，如图 4.18 所示，在该页面就可以查看相应的目录。

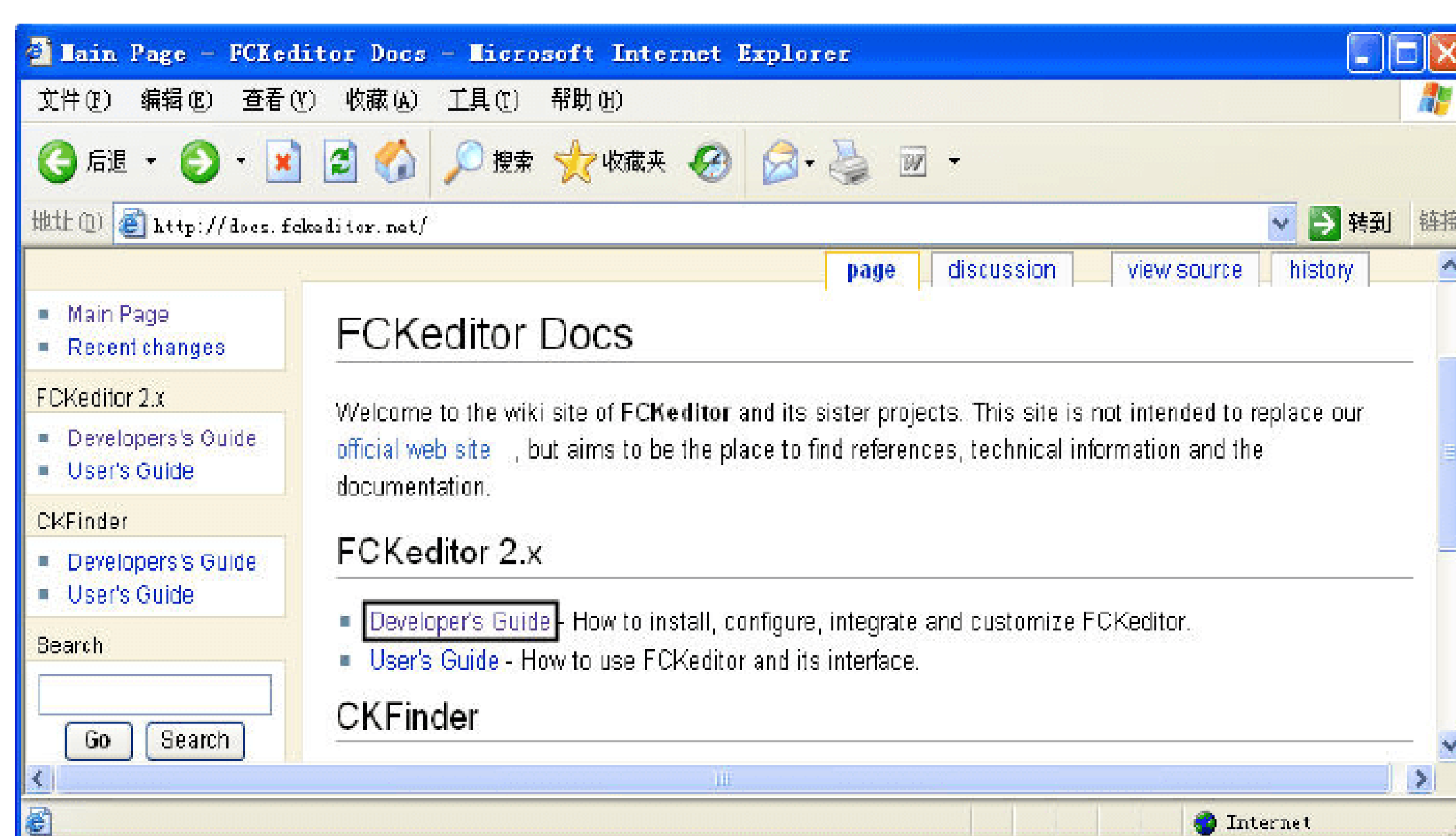


图 4.17　开发文档

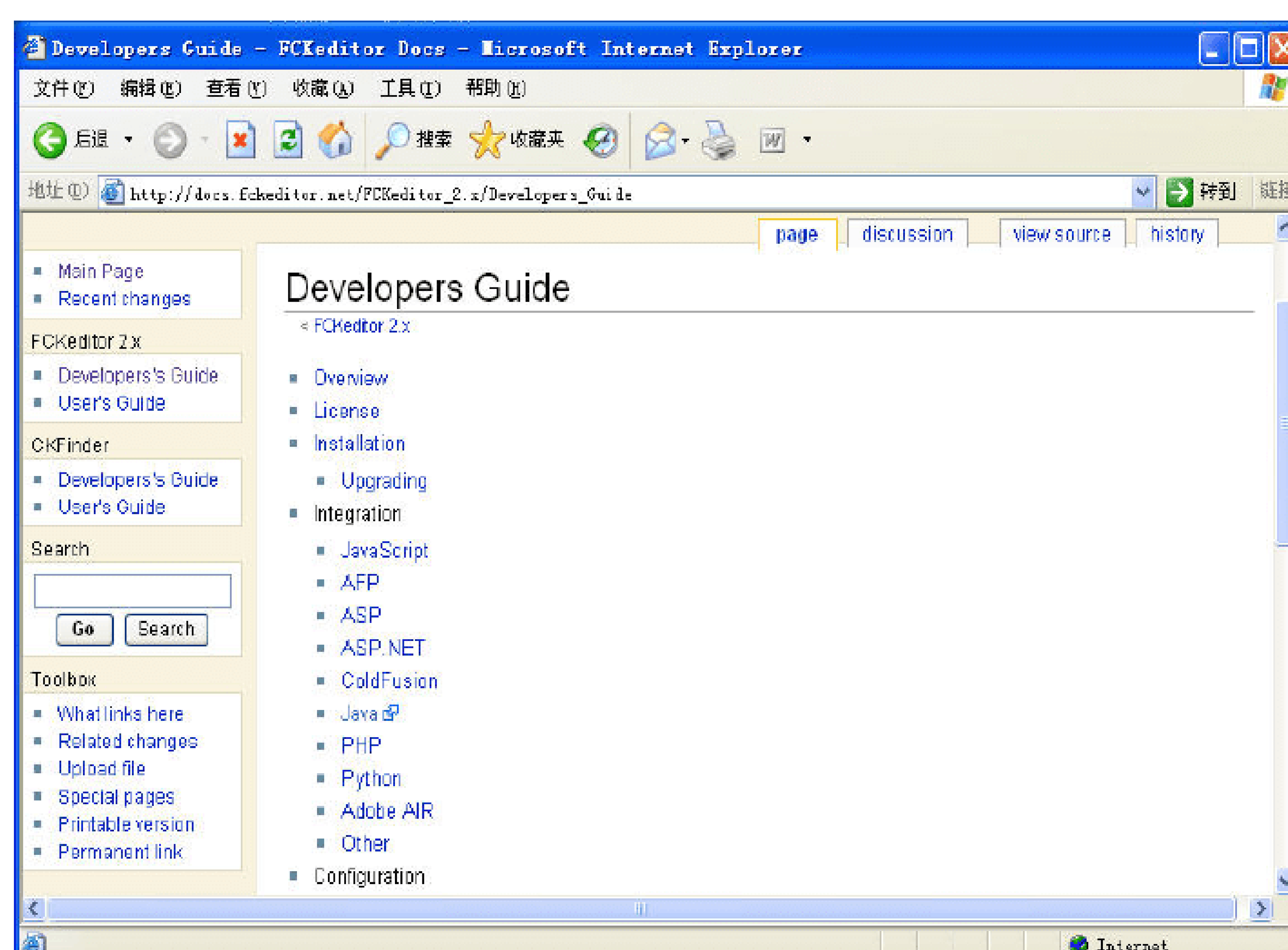


图 4.18　开发文档目录

如果想查看开发实例，可以通过浏览器浏览\fckeditor_samples\default.html 页面来打开实例，如图 4.19 所示。对于开发连接服务器的 FCKeditor 在线文本编辑器，可以通过访问 fckeditor-java-2.5-bin\fckeditor-java-2.5\site\index.html 页面来打开帮助文档，如图 4.20 所示。

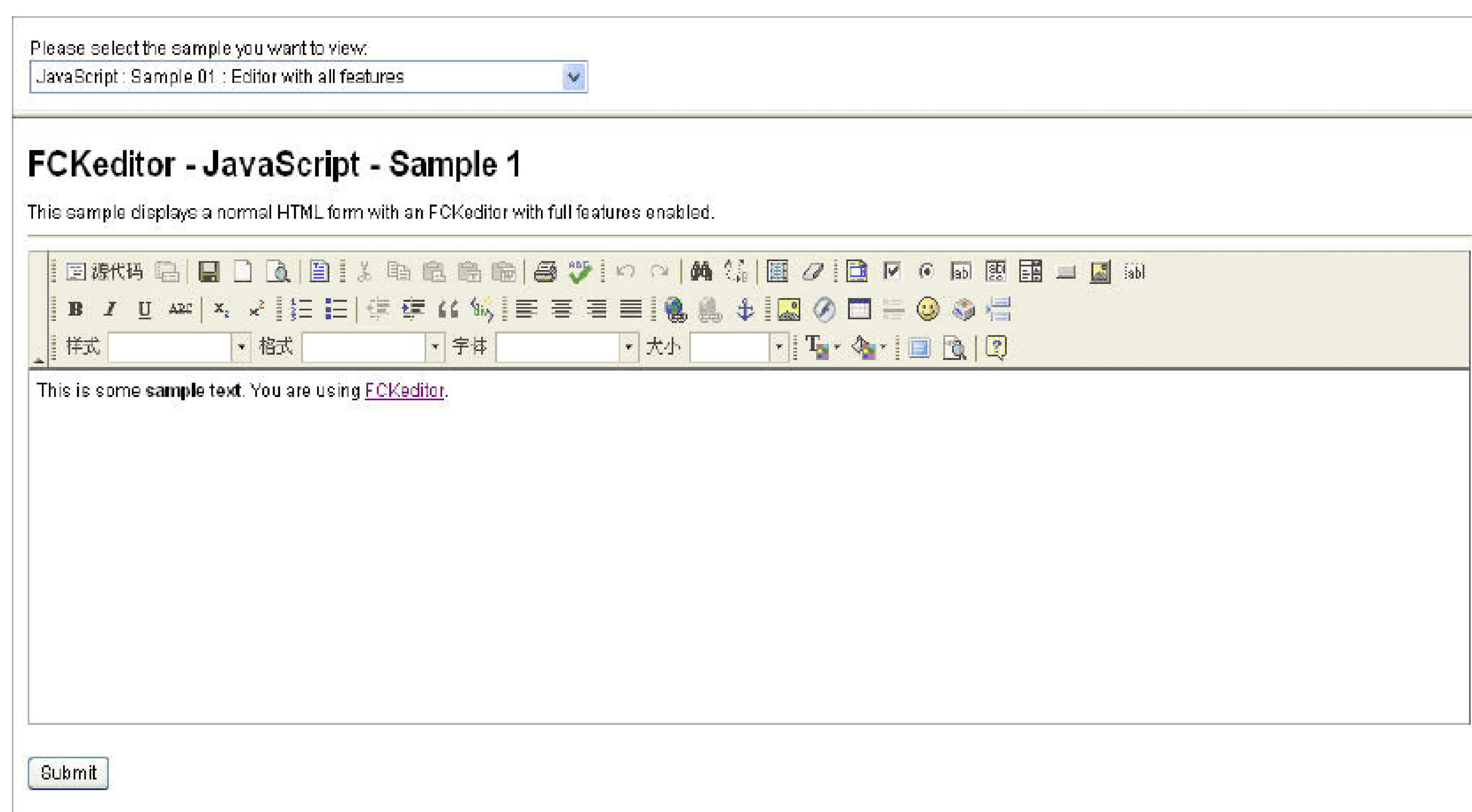


图 4.19　开发实例



图 4.20　开发文档

4.2　FCKeditor 在线文本编辑器初级应用

FCKeditor 在线文本编辑器经常会出现在 BS 系统中的网页中，这些页面经常会用来实现浏览者信息的收集、新闻内容的显示等。在本节中将介绍如何在表单页面中引入 FCKeditor 在线文本编辑器，以及如何实现对 FCKeditor 在线文本编辑器的配置。

4.2.1　利用 JavaScript 语言调用 FCKeditor 在线文本编辑器

如何将 FCKeditor 在线文本编辑器嵌入自己编写的页面？阅读开发文档知道，可以通过两种方式在页面中调用 FCKeditor 在线文本编辑器。在具体实现之前，先配置一下编程环境。

新建一个名叫 FCKeditortest 的 JavaWeb 项目，然后把解压后的 fckeditor 文件夹及该文件夹下的所有文件复制到 FCKeditortest/WebRoot 目录下，最后目录结果如图 4.21 所示。

上述目录之所以会出现红叉，不是因为 fckeditor 文件夹中的代码存在错误，而是有一些代码没有编写完。为了消除红叉，可以取消 MyEclipse 开发环境的验证功能。即通过右击该项目，在弹出的快捷菜单中选择 MyEclipse|Exclude From Validation 命令（如图 4.22 所示），实现相应功能。



图 4.21　目录结构

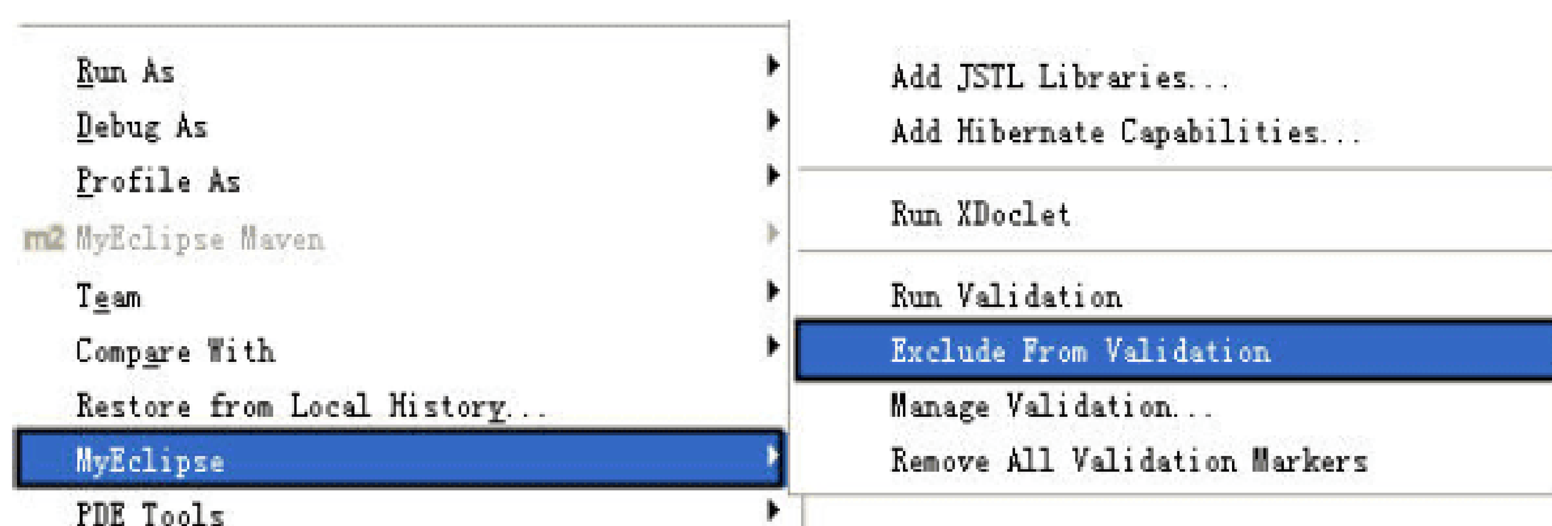


图 4.22　取消验证

至此，就完成对 FCKeditor 在线文本编辑器环境的配置。接着就在 FCKeditortest 项目中通过两种方式实现调用 FCKeditor 在线文本编辑器。

1. JavaScript语言直接调用FCKeditor

当通过 JavaScript 语言直接调用 FCKeditor 在线文本编辑器时，需要新建一个名叫 JavaScript1.jsp 的 JSP 页面，代码 4.1 实现了对 FCKeditor 在线文本编辑器的调用。

代码 4.1　调用在线文本编辑器：JavaScript1.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<html>
    <head>
        <title>通过 JavaScript 语言调用</title>
        <!--引入外部的 js 文件-->
        <script type="text/javascript" src="fckeditor/fckeditor.js">
        </script>
    </head>
    <body>
        通过 JavaScript 语言调用.
        <br>
        <!--调用 FCKeditor 在线文本编辑器-->
        <script type="text/javascript">
        var oFCKeditor = new FCKeditor('FCKeditor1');
        oFCKeditor.BasePath = "/FCKeditortest/fckeditor/";
        oFCKeditor.Create();
        </script>
    </body>
</html>
```

【代码解析】

在引入外部 JavaScript 文件时，src 属性值使用的是相对路径地址。在调用 FCKeditor 在线文本编辑器的 JavaScript 语句中：第 1 句新建一个名为 oFCKeditor 的 FCKeditor 对象、第 2 句用来设置在线文本编辑器的基准路径、第 3 句通过调用 Create()方法来创建和输出一个在线文本编辑器。

注意：属性 basePath 的值代表的是 fckeditor.js 的目录，其“/FCKeditortest/fckeditor/”代码中的第一个“/”代表当前站点的目录，所以还需要加入项目的工程名。

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/FCKeditortest/JavaScript1.jsp，运行结果如图 4.23 所示。

2．标签<textarea>调用FCKeditor

当利用 JavaScript 语言中替代标签<textarea>，实现对 FCKeditor 在线文本编辑器的调用时，需要新建一个名叫 JavaScript2.jsp 的 JSP 页面，代码 4.2 实现对 FCKeditor 在线文本编辑器的调用。

代码 4.2　调用在线文本编辑器：JavaScript2.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<html>
    <head>
        <title>通过 JavaScript 语言调用</title>
        <!--引入外部的 JS 文件-->
        <script type="text/javascript" src="fckeditor/fckeditor.js">
        </script>
        <!--引入网页后，需要加载的代码-->
        <script type="text/javascript">
        window.onload = function()
        {
            var oFCKeditor = new FCKeditor( 'MyTextarea' ) ;
            oFCKeditor.BasePath = "/FCKeditortest/fckeditor/" ;
            oFCKeditor.ReplaceTextarea() ;
        }
        </script>
    </head>
    <body>
        通过 JavaScript 语言调用
        <textarea name="MyTextarea">测试语言</textarea>
    </body>
</html>
```

【代码解析】

在引入外部 JavaScript 文件后，需要定义一个方法，该方法为 window.onload 属性的值，即当浏览器打开网页后就会加载该方法中的代码。在该方法中，第 1 句新建一个名为 oFCKeditor 的 FCKeditor 对象；第 2 句用来设置在线文本编辑器的基准路径，其值除了上述路径外，还可以设置成相对路径即“fckeditor/”；第 3 句通过 ReplaceTextarea()方法替换<textarea>元素。最后还需要配置<textarea>元素，在该元素中必须设置 name 属性。

注意：新建 FCKeditor 对象时，其传递过来的参数必须是标签<textarea>元素的 name 值。

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/FCKeditortest/JavaScript2.jsp，运行结果如图 4.24 所示。

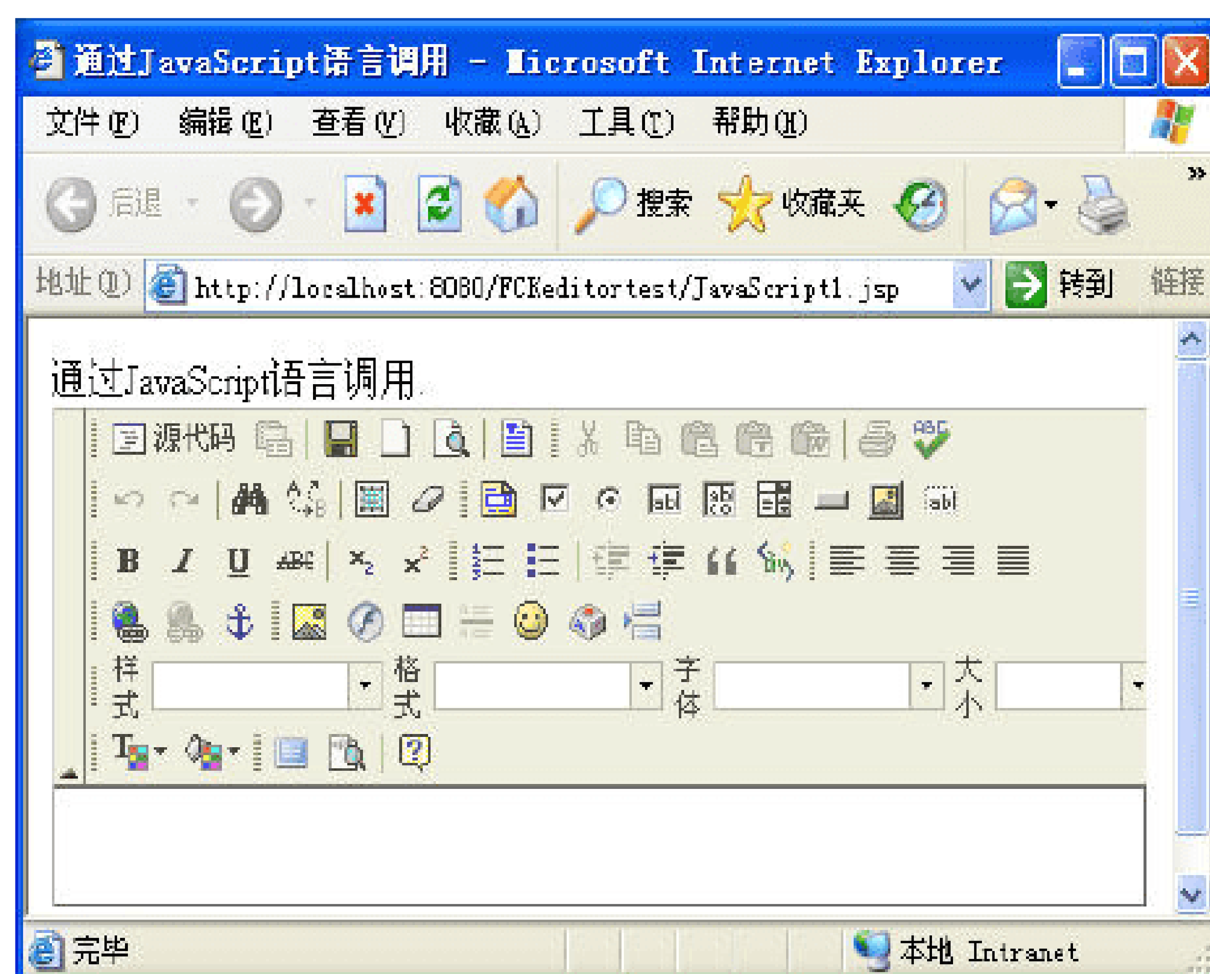


图 4.23　JavaScript1 页面

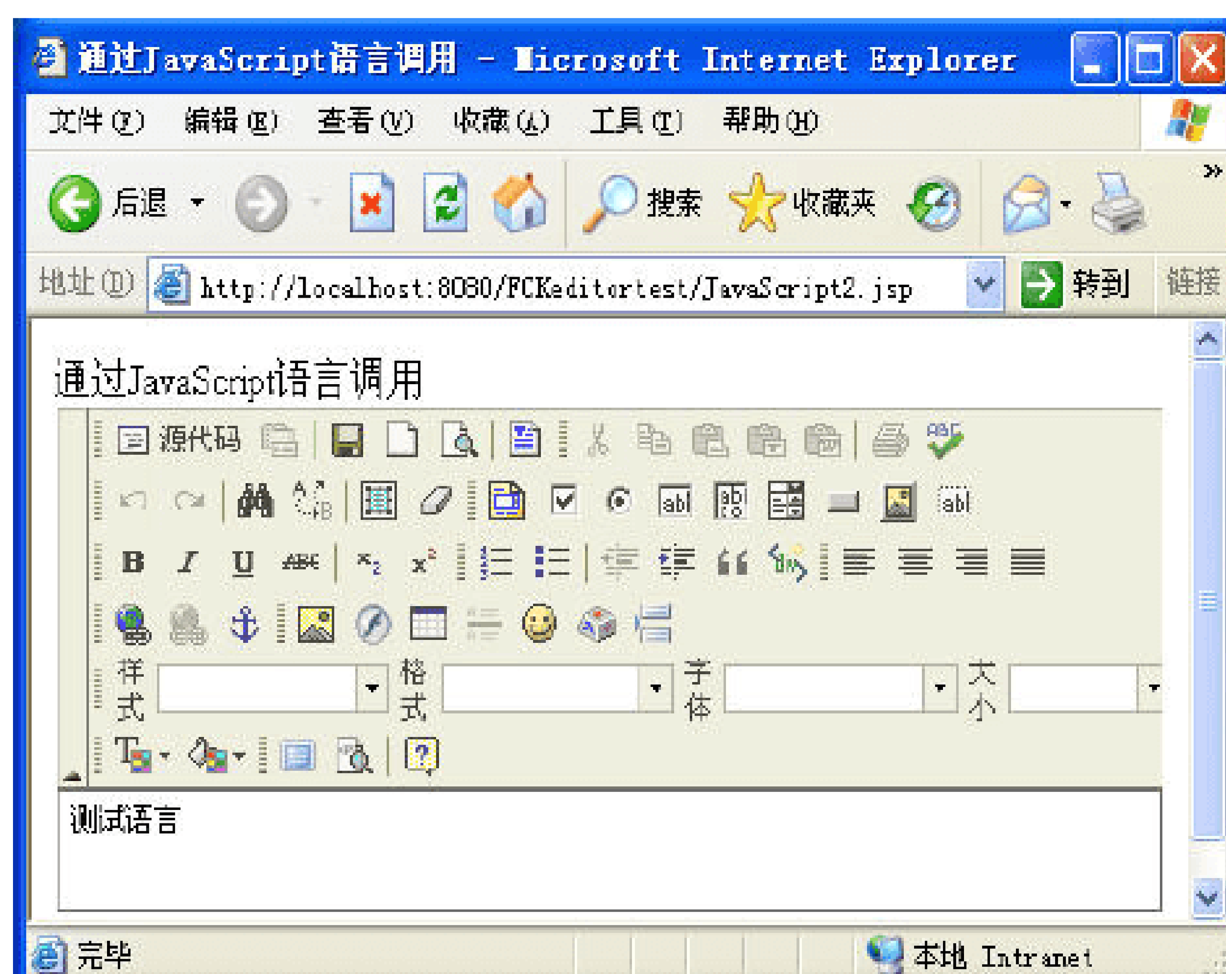


图 4.24　JavaScript2 页面

4.2.2　利用 JSP 标签调用 FCKeditor 在线文本编辑器

除了可以利用 JavaScript 语言在页面中调用 FCKeditor 在线文本编辑器外，还可以通过使用 JSP 标签来实现对 FCKeditor 在线文本编辑器的调用，本节将详细介绍该种方式。

由于 JSP 标签是服务器端编程技术，在具体使用时需要连接服务器，所以需要引入 fckeditor-java-2.4.1-bin.zipjar 文件中的类来配置开发环境。具体配置步骤如下。

（1）首先把 fckeditor-java-2.4.1\lib 目录下的 3 个 jar 文件：commons-fileupload-1.2.1.jar、commons-io-1.3.2.jar、slf4j-api-1.5.2.jar 和解压后的 fckeditor-java-core-2.4.1jar 文件（共 4 个 jar 文件），引入 Java Web 项目。

（2）当只引入这 4 个 jar 文件后，在测试项目时会发生如图 4.25 所示的错误。这主要是没引入 slf4j-simple-1.5.2.jar 相关 jar 文件，可以通过解压 fckeditor-java-demo-2.4.1.war 测试项目，从其目录 fckeditor-java-demo-2.4.1\WEB-INF\lib 中找到它。

至此，就完成对 FCKeditor 在线文本编辑器环境的配置。接着编写一个名为 JSP 的文件，代码 4.3 实现通过 JSP 页面中的自定义标签来对 FCKeditor 在线文本编辑器进行调用。

代码 4.3　调用在线文本编辑器：JSP.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<!--引入标签库-->
<%@ taglib uri="http://java.fckeditor.net" prefix="FCK" %>>
...
    <body>
        通过 JSP 页面中的自定义标签来调用 FCKeditor 在线文本编辑器
        <!--设置 FCK 标签-->
        <FCK:editor instanceName="editorDefault"  basePath="/fckeditor"
        value="测试代码"></FCK:editor>
    </body>
</html>
```

【代码解析】

在编写<FCK>标签时，其属性 basePath 的值必须以“/”开始，代表当前项目的根目录。同时还要注意设置 value 的属性值，并且值不能是空字符串。

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/FCKedi-

tortest/JSP.jsp，运行结果如图 4.26 所示。

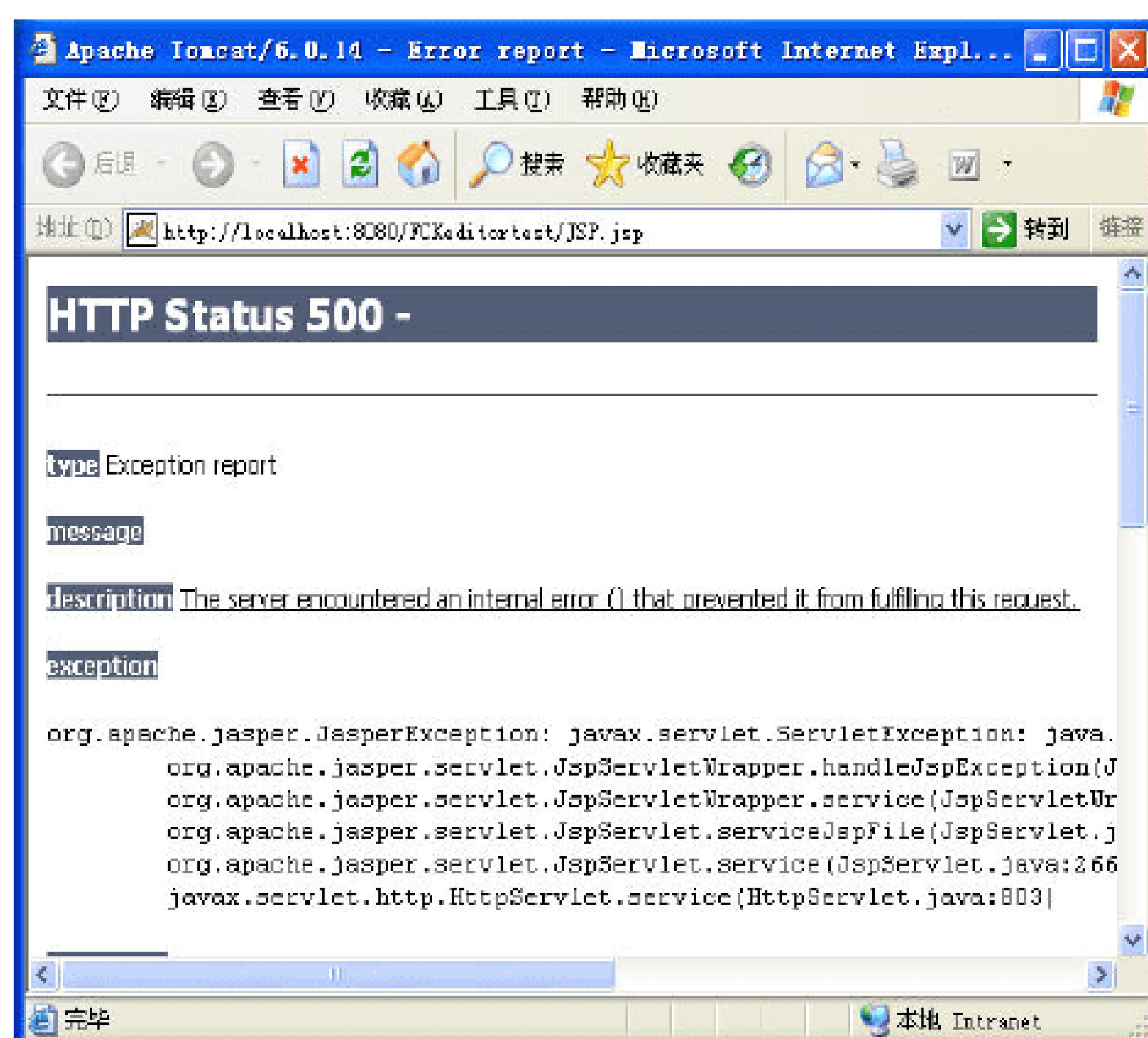


图 4.25　发生异常

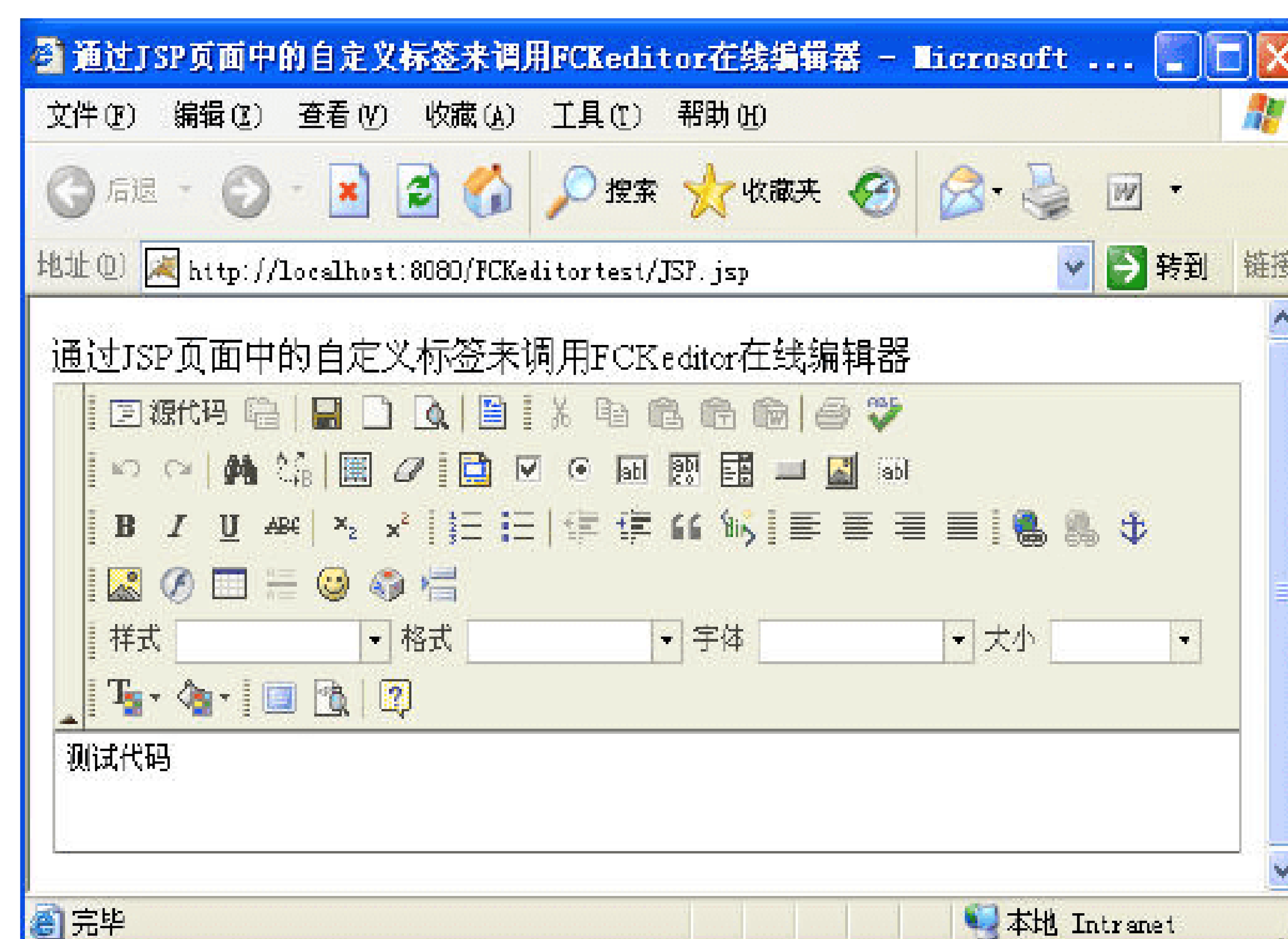


图 4.26　JSP 页面

4.3　FCKeditor 在线文本编辑器常用配置

如果想让 FCKeditor 在线文本编辑器更便于用户使用，除了把其引入表单页面外，还要对其进行必要的配置。因为默认的 FCKeditor 在线文本编辑器存在许多弊端（如图 4.27 所示），所以一般需要自定义编辑器的工具栏、设置编辑器的常用字体、设置编辑器的表情图像和修改 Enter 和 Shift+Enter 键的行为。



图 4.27　默认的功能

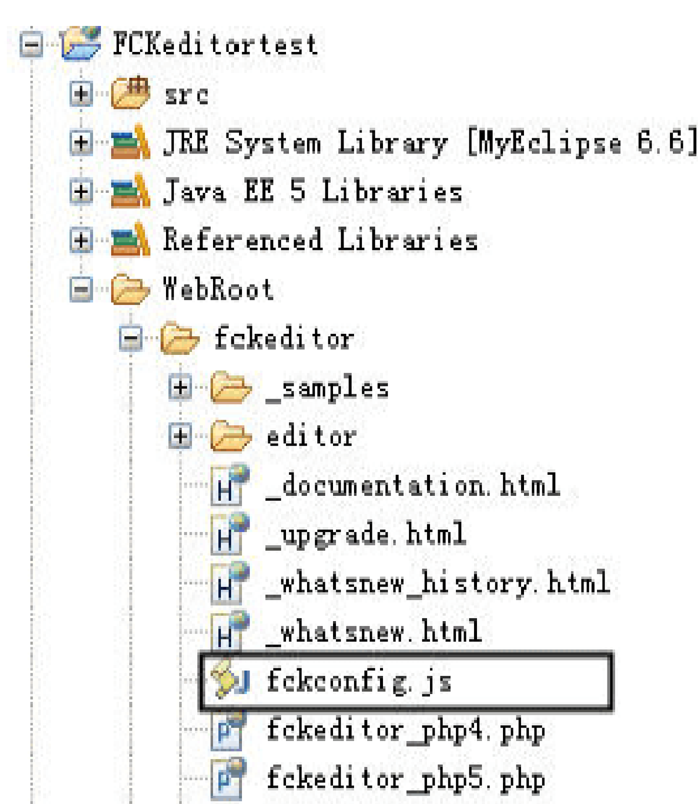
4.3.1　修改配置文件

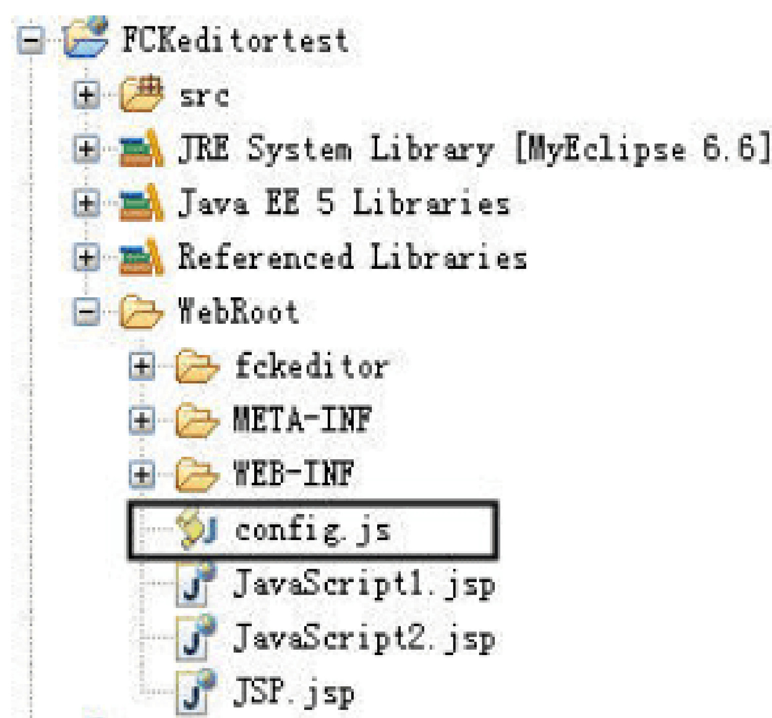
如何修改 FCKeditor 在线文本编辑器的默认配置呢？一般只需要修改该编辑器的配置文件就可以。那如何修改配置文件呢？本节将详细介绍关于配置文件方面的知识。

FCKeditor 在线文本编辑器有一个主配置文件，对于项目 FCKeditortest 来说其路径如图 4.28 所示。所谓的配置 FCKeditor 在线文本编辑器，就是修改 FCKeditor 类的 fckconfig 属性，而该属性的各个选项就都存储于 fckconfig.js 配置文件中。

根据开发文档可以知道，在配置 FCKeditor 在线文本编辑器时，一般不会修改主配置文件，而是创建一个新的配置文件来覆盖主配置文件。在该新的配置文件中只需要编写需要修改的配置项就可以。因此创建一个名为 config.js 的配置文件，那么新配置文件的位置如何选择呢，是 FCKeditortest/WebRoot 目录下（如图 4.29 所示）还是 FCKeditortest/WebRoot/

fckeditor 目录下（如图 4.30 所示）？





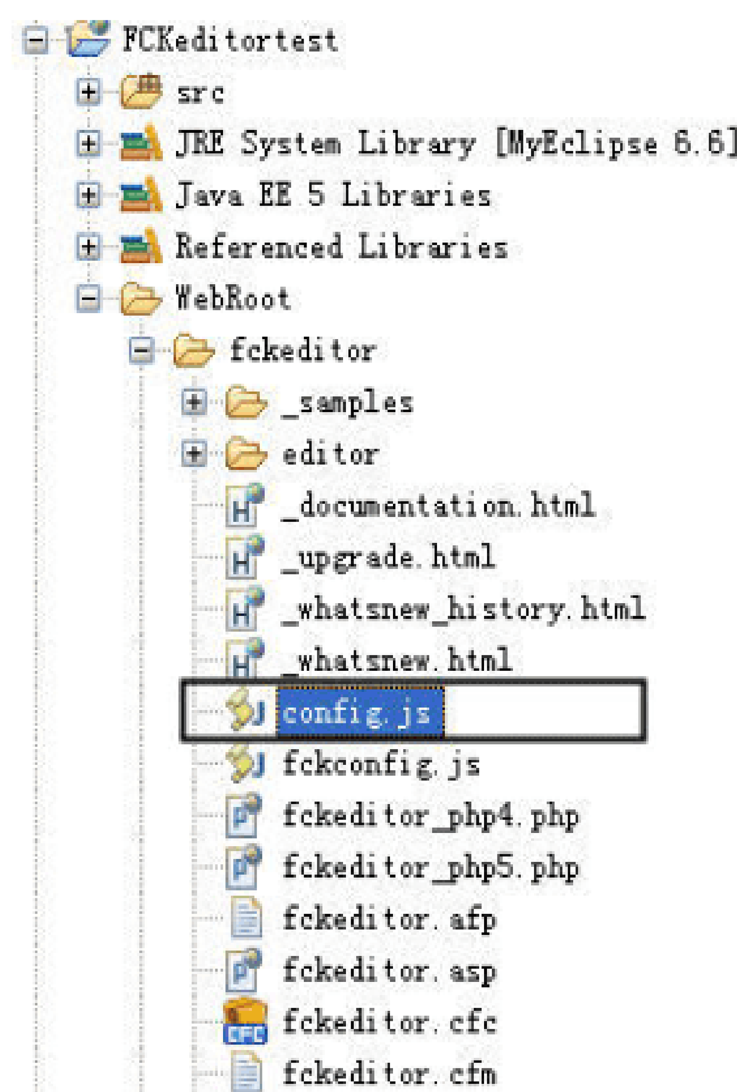


图 4.28　主配置文件　　　图 4.29　存放路径 1　　　图 4.30　存放路径 2

对于上述第一种方案，如果想表示 config.js 文件的路径，必须用路径 FCKeditortest/config.js 表示。对于上述第二种方案，如果想表示 config.js 文件的目录，除了可以用路径 FCKeditortest/fckeditor/config.js 表示外，还可以通过路径 FCKConfig.EditorPath/config.js 表示。

注意：由于项目的名称在发布之前是不确定的，所以第一种方案不如第二种方案合适。

修改配置文件的具体步骤如下。

（1）通过网址 http://docs.fckeditor.net/FCKeditor_2.x/Developers_Guide/Configuration/Configuration_File 打开配置文档的帮助文档页面，首先复制该页面中步骤一的第一行代码到 config.js 文件中，具体内容如下所示。

```
FCKConfig.AutoDetectLanguage = false ;
```

属性 AutoDetectLanguage 用来配置 FCKeditor 在线文本编辑器是否自动检查语言，该属性的默认值为 true，表示在使用该编辑器时自动检查语言。

（2）如何使用新的配置文件呢？可以有两种方案：在主配置文件中配置或在调用编辑器的页面中配置。

1. 配置主配置文件

如果想配置主配置文件，可以打开 fckconfig.js 配置文件，修改第一行代码中 CustomConfigurationsPath 属性的值为 config.js 的路径。

```
FCKConfig.CustomConfigurationsPath = '' ;
```

注意：第一个“/”代表当前站点目录。

当浏览 FCKeditortest 项目中的 JavaScript1.jsp 页面、JavaScript2.jsp 或 JSP.jsp 页面时，FCKeditor 在线文本编辑器如图 4.31 所示，显示的都是英文。

2. 配置调用页面

如果想配置调用页面，则需要在每个页面中调用新的配置文件方案，代码 4.4 通过调

用 config.js 文件覆盖主配置文件中相同的选项。

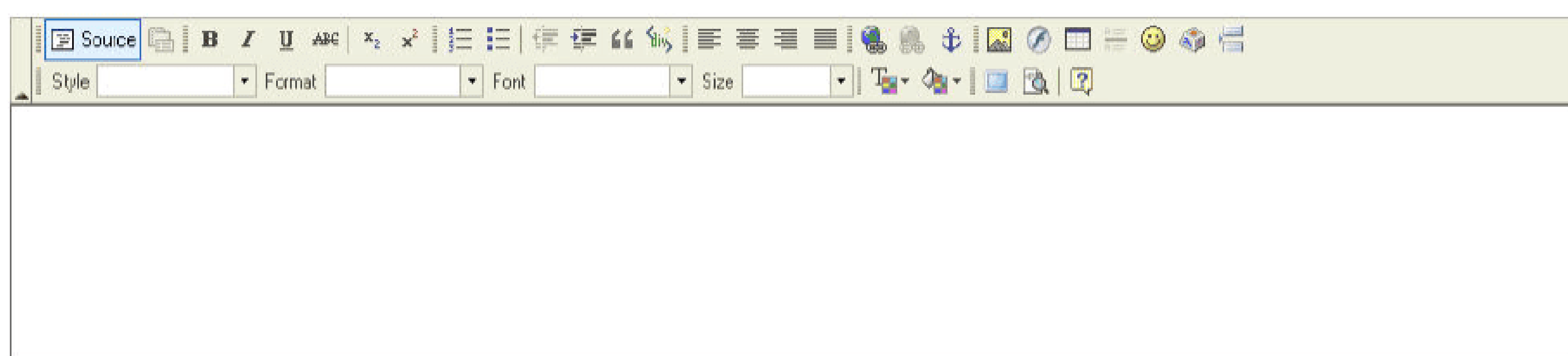


图 4.31　FCKeditor 在线文本编辑器

代码 4.4　调用新的配置文件：JavaScript1.jsp

```
...
    <body>
        通过 JavaScript 语言调用.
        <script type="text/javascript">
        var oFCKeditor = new FCKeditor('FCKeditor1');
        oFCKeditor.BasePath = "/FCKeditortest/fckeditor/";
        //配置 CustomConfigurationsPath 属性
        oFCKeditor.Config["CustomConfigurationsPath"] = "/FCKeditortest/
        fckeditor/config.js" ;
        oFCKeditor.Create();
        </script>
    </body>
...
```

【代码解析】

- ❑ 属性 Config["CustomConfigurationsPath"]的值为 config.js 文件的目录地址，在这里因为 config.js 文件存放在 FCKeditortest/fckeditor 目录下，所以其值为/FCKeditortest/fckeditor/config.js，最前面的“/”表示当前站点目录。
- ❑ 当浏览 FCKeditortest 项目中的 JavaScript1.jsp 页面时，会显示如图 4.31 所示的英文 FCKeditor 在线文本编辑器。可是当浏览 JavaScript2.jsp 或 JSP.jsp 页面时，显示的却是中文 FCKeditor 在线文本编辑器。
- ❑ 通过两种方案的比较可以知道，第一种方案对所有的 FCKeditor 在线文本编辑器实例都有效，而第二种方案只对调用新配置文件的实例有效。

注意：*在修改配置后，如果要查看运行结果，必须清空浏览器。清空浏览器时，对于 IE 浏览器可以使用 Ctrl+F5 快捷键，而对于 Firefox 浏览器却是使用 Shift+Ctrl+R 快捷键。*

通过上述实例可以知道，当项目在加载配置文件时，首先加载主配置文件 fckconfig.js，然后加载自定义的配置文件，覆盖相同的配置项。

4.3.2　自定义工具栏

FCKeditor 在线文本编辑器工具栏中的好多工具都不需要，所以在具体开发时都需要修改工具栏，即去掉一些不必要的工具，在本节中将详细介绍如何自定义工具栏。具体步

骤如下。

（1）首先编写 config.js 文件，修改 FCKConfig 类的 ToolbarSets 属性来实现自定义工具栏。代码 4.5 为自定义工具栏的配置文件代码。

代码 4.5　自己配置文件：config.js

```
//定制工具栏
FCKConfig.ToolbarSets["myself"] = [
    ['Source','DocProps'],
    ['Bold','Italic','Underline','StrikeThrough','-','Subscript','Supers
    cript'],
    ['OrderedList','UnorderedList','-','Outdent','Indent','Blockquote',
    'CreateDiv'],
    ['JustifyLeft','JustifyCenter','JustifyRight','JustifyFull'],
    ['Link','Unlink','Anchor'],
    ['Image','Flash','Table','Rule','Smiley','SpecialChar','PageBreak'],
    '/',
    ['Style','FontFormat','FontName','FontSize'],
    ['TextColor','BGColor'],
    ['FitWindow','ShowBlocks','-','About']
] ;
```

【代码解析】

FCKConfig.ToolbarSets["myself"]代码中的 myself 为自定义工具条的名称，其中属性值中的“[”和“]”用来设置▏，使用其隔开工具栏中不同用途类型的按钮；属性值中的“-”用来设置▏，使用其隔开工具栏中同用途类型的按钮；属性值中的“/”用来设置换行，使总的工具栏按钮分成两行；而其他属性值可以与默认编辑器中的工具栏按钮相对应。例如 Source 用来设置 源代码，DocProps 用来设置 等。

（2）在 JavaScript1.jsp 页面中调用新的配置文件 config.js，同时设置工具栏的名称为自定义工具栏名称 myself，代码 4.6 用来实现调用配置文件。

代码 4.6　调用配置文件：JavaScript1.jsp

```
...
    <body>
        通过 JavaScript 语言调用.
        <br>
        <script type="text/javascript">
        var oFCKeditor = new FCKeditor('FCKeditor1');
        oFCKeditor.BasePath = "/FCKeditortest/fckeditor/";
        //配置 CustomConfigurationsPath 属性
        oFCKeditor.Config["CustomConfigurationsPath"] = "/FCKeditortest/
        config.js" ;
        //设置属性 ToolbarSet
        oFCKeditor.ToolbarSet="myself";
        oFCKeditor.Create();
        </script>
    </body>
...
```

【代码解析】

上述代码中属性 ToolbarSet 的值为自定义工具栏的名称。单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在

地址栏中输入地址 http://localhost:8080/FCKeditortest/JavaScript1.jsp，运行结果如图 4.32 所示。

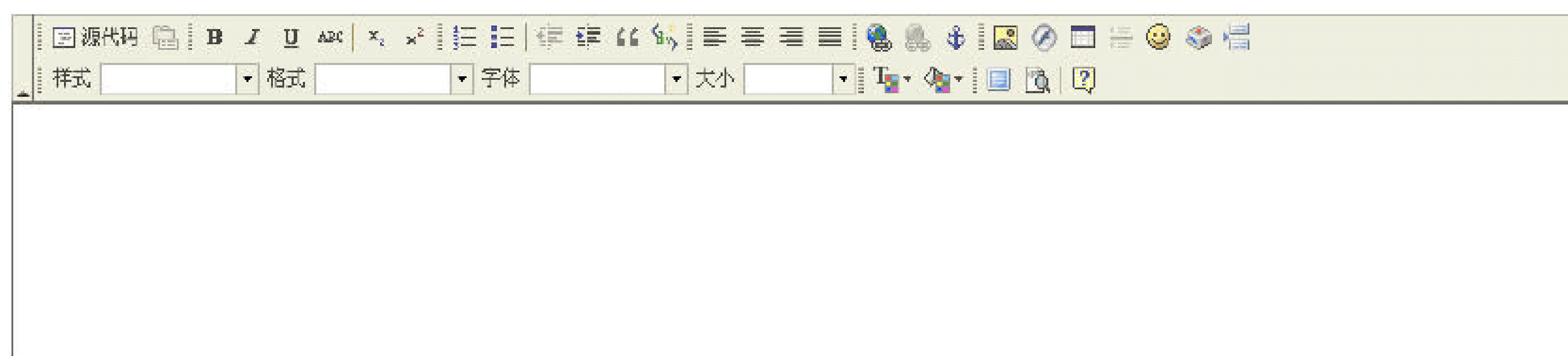


图 4.32　修改后的工具栏

4.3.3　设置常用的字体和键行为

FCKeditor 在线文本编辑器中除了常用的字体有问题外，并且 Enter 和 Shift+Enter 键的行为正好相反，所以需要对常用字体和一些键的行为进行设置和修改，使浏览者更容易使用。具体步骤如下。

（1）首先编写 config.js 文件，修改 FCKConfig 类的 FontNames 属性来设置常用字体；修改 FCKConfig 类的 EnterMode 属性设置 Enter 键的行为；修改 FCKConfig 类的 ShiftEnterMode 属性设置 Shift+Enter 键的行为。代码 4.7 为配置文件代码。

代码 4.7　配置文件：config.js

```
//设置常用字体
FCKConfig.FontNames= '宋体;楷体_GB2312;黑体;Times New Roman;Verdana' ;
//交互 Enter 和 Shift+Enter 的行为
FCKConfig.EnterMode = 'br' ;
FCKConfig.ShiftEnterMode = 'p' ;
```

当在保存该文件时会出现如图 4.33 所示的错误，这是因为在设置常用字体时使用了中文。如果想解决该问题，首先必须关闭该页面，然后在 Package Explorer 视图中右击 config.js 文件，在出现的快捷菜单中选择 Properties 属性，弹出如图 4.34 所示的对话框。在该对话框中的 Text file encoding 选项区域中选择 Other 单选按钮，并在后面的下拉列表框中选中 UTF-8 编码格式。

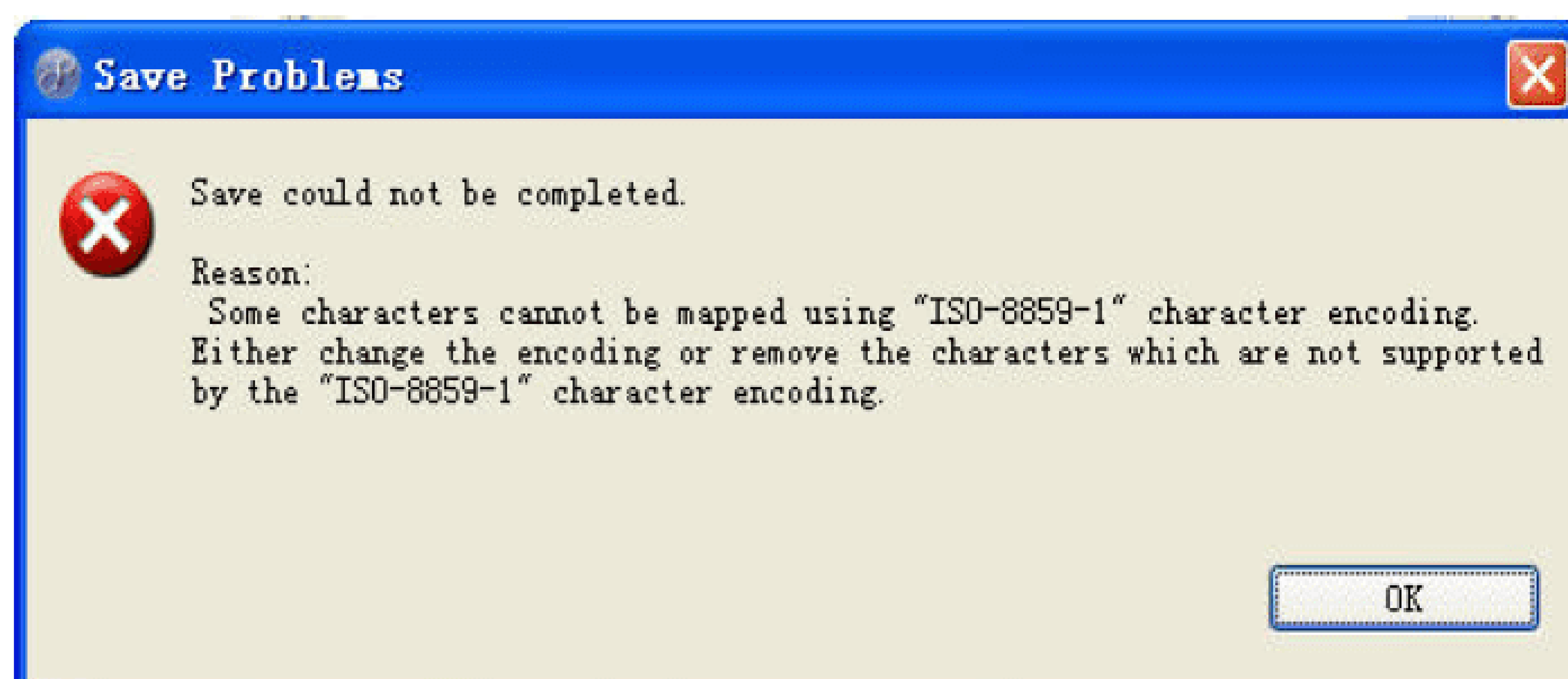


图 4.33　保存错误

（2）在 JavaScript1.jsp 页面中调用新的配置文件 config.js。

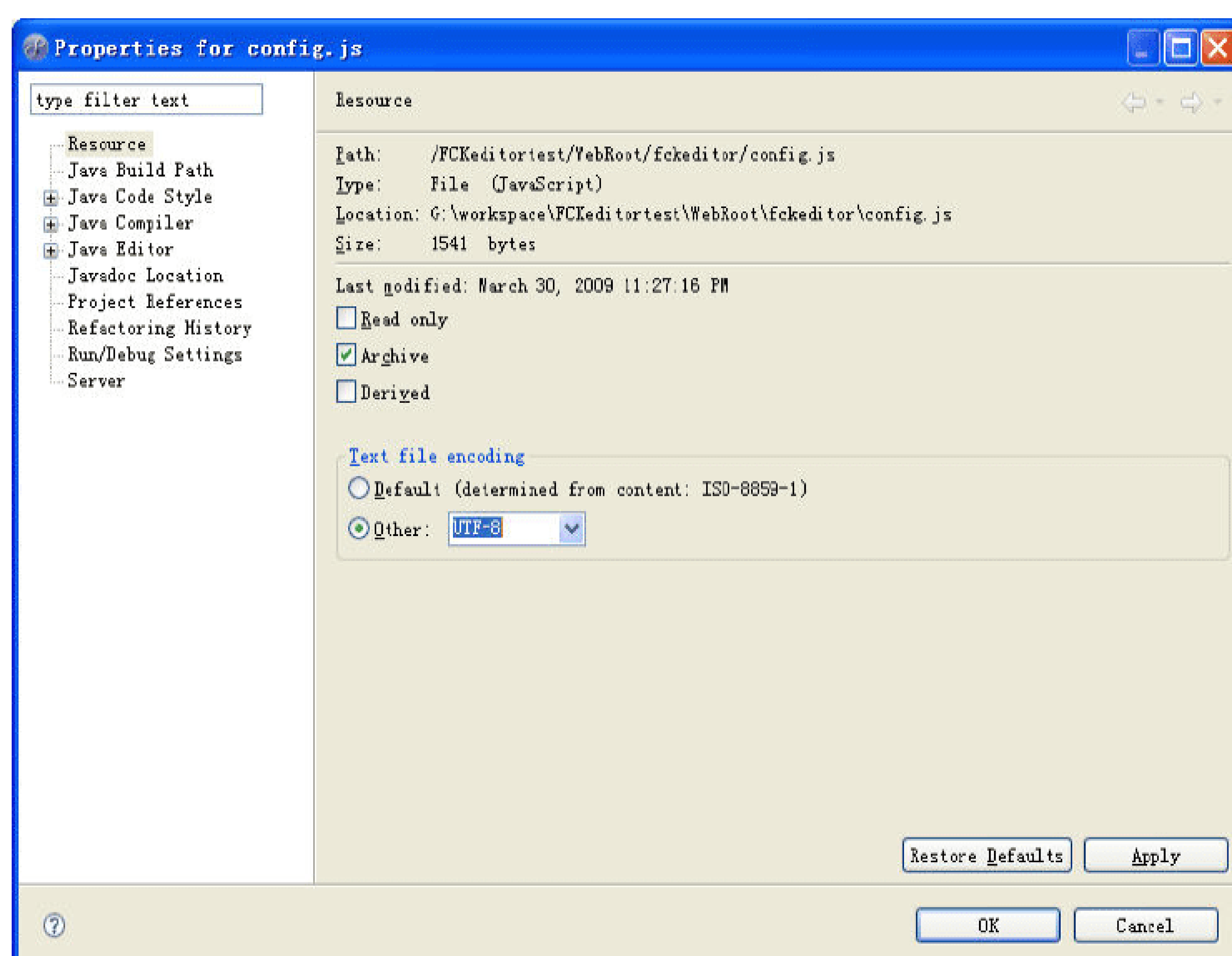


图 4.34　属性对话框

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/FCKeditortest/JavaScript1.jsp，运行结果如图 4.35 所示。

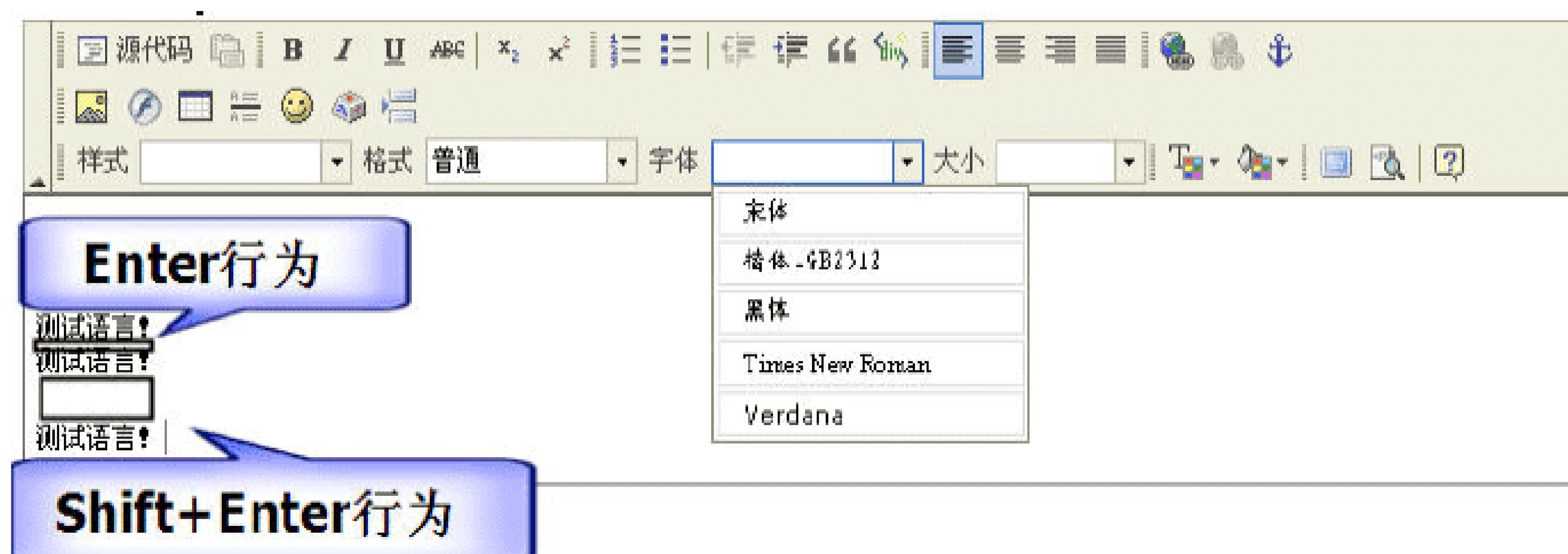


图 4.35　修改后的编辑器

4.3.4　修改插入表情图标

为了扩充表情图标内容，可以修改 FCKeditor 在线文本编辑器中的插入表情图标。在本节中将详细介绍如何设置插入表情图标。具体步骤如下。

（1）首先编写 config.js 文件，修改 FCKConfig 类的 5 个属性：SmileyPath、SmileyImages、SmileyColumns、SmileyWindowWidth 和 SmileyWindowHeight 来设置 FCKeditor 在线文本编辑器的插入表情图标，代码 4.8 为配置文件代码。

代码 4.8　配置文件：config.js

```
//设置表情图标的路径
FCKConfig.SmileyPath= FCKConfig.BasePath + 'images/smiley/msn/' ;
//设置所要使用的表情图标的名称
FCKConfig.SmileyImages  =
```

```
['gougou.gif','regular smile.gif','sad smile.gif','wink smile.gif','tee
th smile.gif','confused smile.gif','tounge smile.gif','embaressed smile
.gif','omg_smile.gif','whatchutalkingabout_smile.gif','angry_smile.gif'
,'angel smile.gif','shades smile.gif','devil smile.gif','cry smile.gif'
,'lightbulb.gif','thumbs down.gif','thumbs up.gif','heart.gif','broken
heart.gif','kiss.gif','envelope.gif'] ;
//设置每行标签图标的个数
FCKConfig.SmileyColumns = 8 ;
//设置显示标签图标窗口的宽度和高度
FCKConfig.SmileyWindowWidth= 320 ;
FCKConfig.SmileyWindowHeight= 210 ;
```

【代码解析】

上述代码中 FCKConfig.SmileyPath 属性的值用来设置表情图标路径；FCKConfig.SmileyImages 属性的值用来设置表情图标的名称；FCKConfig.SmileyColumns 属性的值用来设置每行标签图标的个数；FCKConfig.SmileyWindowWidth 属性的值用来设置显示标签图标窗口的宽度；FCKConfig.SmileyWindowHeight 属性的值用来设置显示标签图标窗口的高度。

如果想查看 FCKConfig.BasePath 属性的路径，可以在 fckconfig.js 文件中输入如下代码：

```
alert(FCKConfig.BasePath );
```

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/FCKeditortest/JavaScript1.jsp，首先会弹出如图 4.36 所示的对话框。

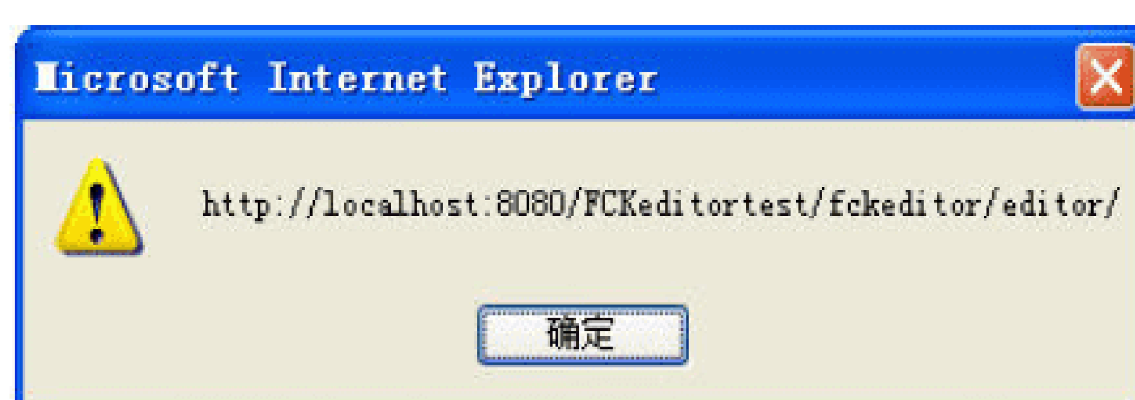


图 4.36　路径

（2）如果想设置在显示表情图标窗口大小的同时显示滚动条，除了设置 FCKConfig.SmileyWindowWidth 和 FCKConfig.SmileyWindowHeight 两属性，还必须修改 fck_smiley.html 文件的代码，如代码 4.9 所示。

代码 4.9　显示滚动条：fck_smiley.html

```
...
</head>
...
<script type="text/javascript">
...
window.onload = function ()
{
    oEditor.FCKLanguageManager.TranslatePage(document) ;
    //dialog.SetAutoSize( true ) ;
}
...
</script>
</head>
```

```
<body style="overflow: srcoll ">
...
</body>
```

【代码解析】

- 在函数 window.onload = function ()中，dialog.SetAutoSize(true)这句代码实现自动调整显示表情图标窗口的大小，去掉这句代码就可以实现对显示窗口大小的调整。
- 在代码<body style="overflow: srcoll ">中，样式表 overflow 的作用是检索和设置当前对象的内容超过其指定高度及宽度时如何管理内容，其有 3 个值，分别为 hidden，不显示超过对象尺寸的内容；auto，自动调整是否显示滚动条和 srcoll，总是显示滚动条。
- 在 JavaScript1.jsp 页面中调用新的配置文件 config.js。

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后单击按钮就会显示出如图 4.37 所示的表情图标窗口。

注意：如何知道修改 fck_smiley.html 文件就可以实现对显示表情图标窗口的修改呢？右击图 4.37 中的图标，在出现的上下文中选择“属性”选项，就可以出现如图 4.38 所示的目录地址。



图 4.37　带有滚动条的窗口



图 4.38　查看处理文件路径

4.4　FCKeditor 在线文本编辑器高级应用

FCKeditor 在线文本编辑器除了可以实现上述几节介绍的功能外，还可以实现创建目录和上传文件、图像、Flash 和视频等功能。本节将详细介绍如何实现上传文件等高级功能，以及在使用这些高级应用时如何解决乱码问题。

4.4.1　FCKeditor 在线文本编辑器上传文件配置

如果想在 FCKeditor 在线文本编辑器中使用上传文件的功能，除了引入 5 个相关 jar

文件外还必须编写服务器端的配置文件，因为该项功能涉及服务器端的编程。首先对 FCKeditortest 项目进行环境的配置，具体步骤如下。

（1）把 commons-fileupload-1.2.1.jar、commons-io-1.3.2.jar、slf4j-api-1.5.2.jar、fckeditor-java-core-2.4.1 和 slf4j-simple-1.5.2.jar 这 5 个相关 jar 文件引入 FCKeditortest 项目。

（2）修改 FCKeditortest/WebRoot/WEB-INF/web.xml 文件，配置实现文件上传功能的类，代码 4.10 实现对上传文件类 ConnectorServlet 的配置。

代码 4.10　配置上传文件类：web.xml

```
<!--配置 ConnectorServlet 类-->
<servlet>
    <servlet-name>Connector</servlet-name>
    <servlet-class>net.fckeditor.connector.ConnectorServlet</servlet-
    class>
    <load-on-startup>1</load-on-startup>
</servlet>
<!--设置 ConnectorServlet 类的映射-->
<servlet-mapping>
    <servlet-name>Connector</servlet-name>
    <url-pattern>
        /fckeditor/editor/filemanager/connectors/*
    </url-pattern>
</servlet-mapping>
```

（3）在目录 FCKeditortest/src 下创建名为 fckeditor 的属性文件。该文件里只需要添加如下一行代码就可以。

```
connector.userActionImpl=net.fckeditor.requestcycle.impl.UserActionImpl
```

注意：该文件的名字必须为 fckeditor.properties，其内容为固定不需要更改。如果想查看帮助文档，可以通过 fckeditor-java-2.4.1-bin\fckeditor-java-2.4.1\site\index.html 目录来实现。

（4）单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮启动服务器，最后打开“图像属性”对话框（如图 4.39 所示），单击“浏览服务器”按钮后，就不会出现如图 4.40 所示的错误。

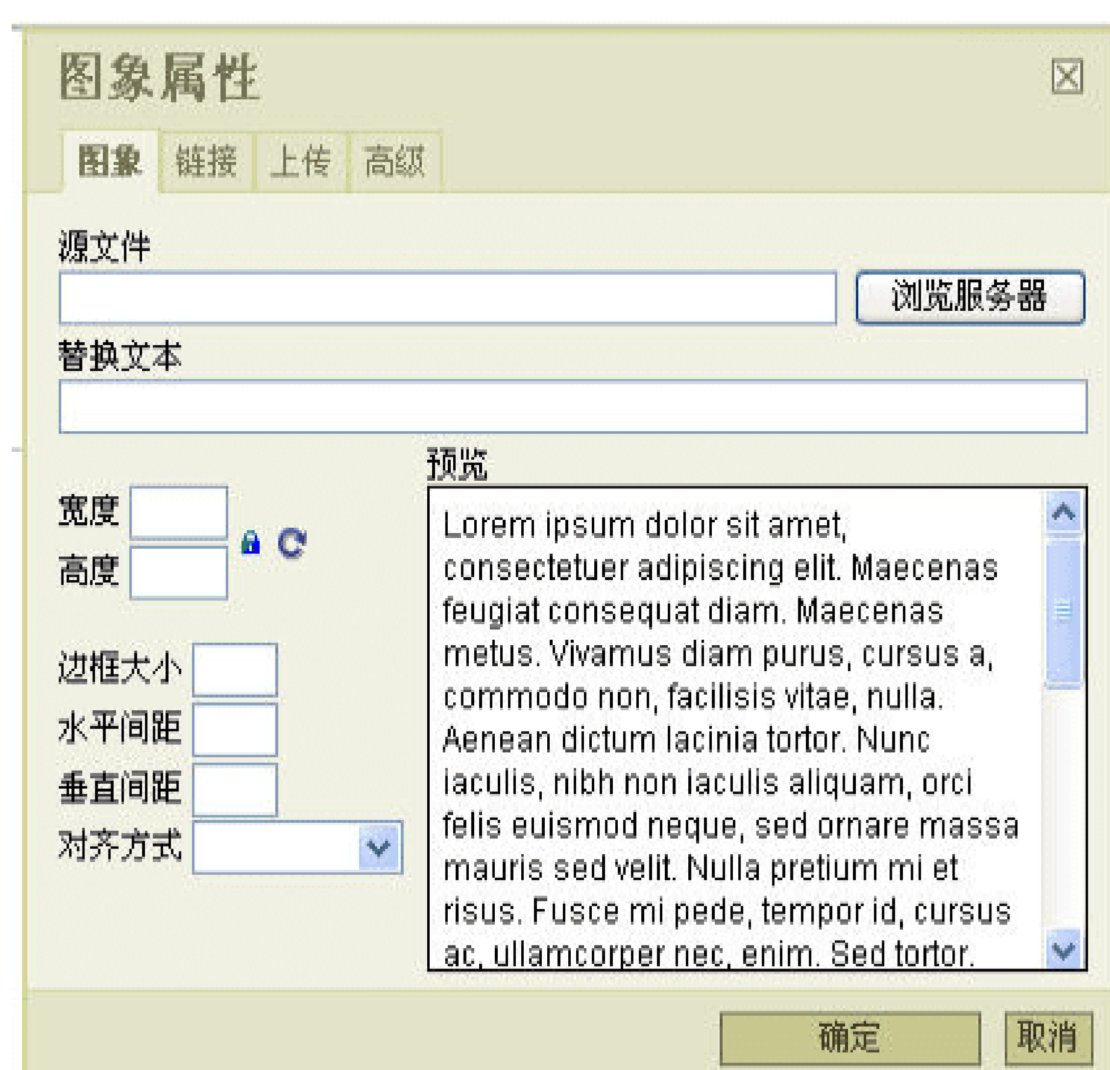


图 4.39　插入图像对话框

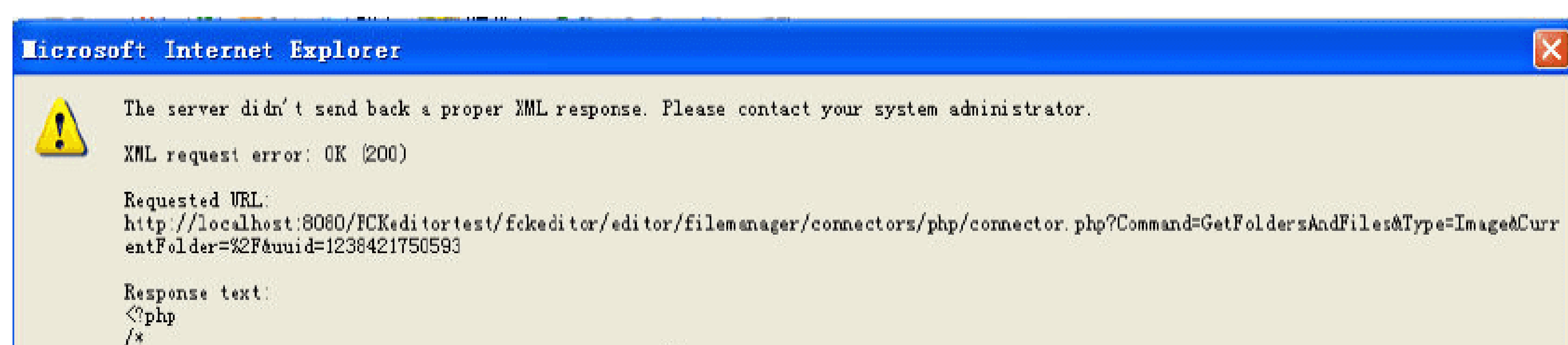


图 4.40　错误

4.4.2　FCKeditor 在线文本编辑器上传文件配置——中文乱码（一）

经过 4.4.1 节的配置，虽然可以实现文件上传功能，但是当上传的文件名为中文（狗.gif）时就会在如图 4.41 所示的上传文件对话框中显示为乱码。之所以会出现这种乱码，可以通过下面两种情况来考虑。

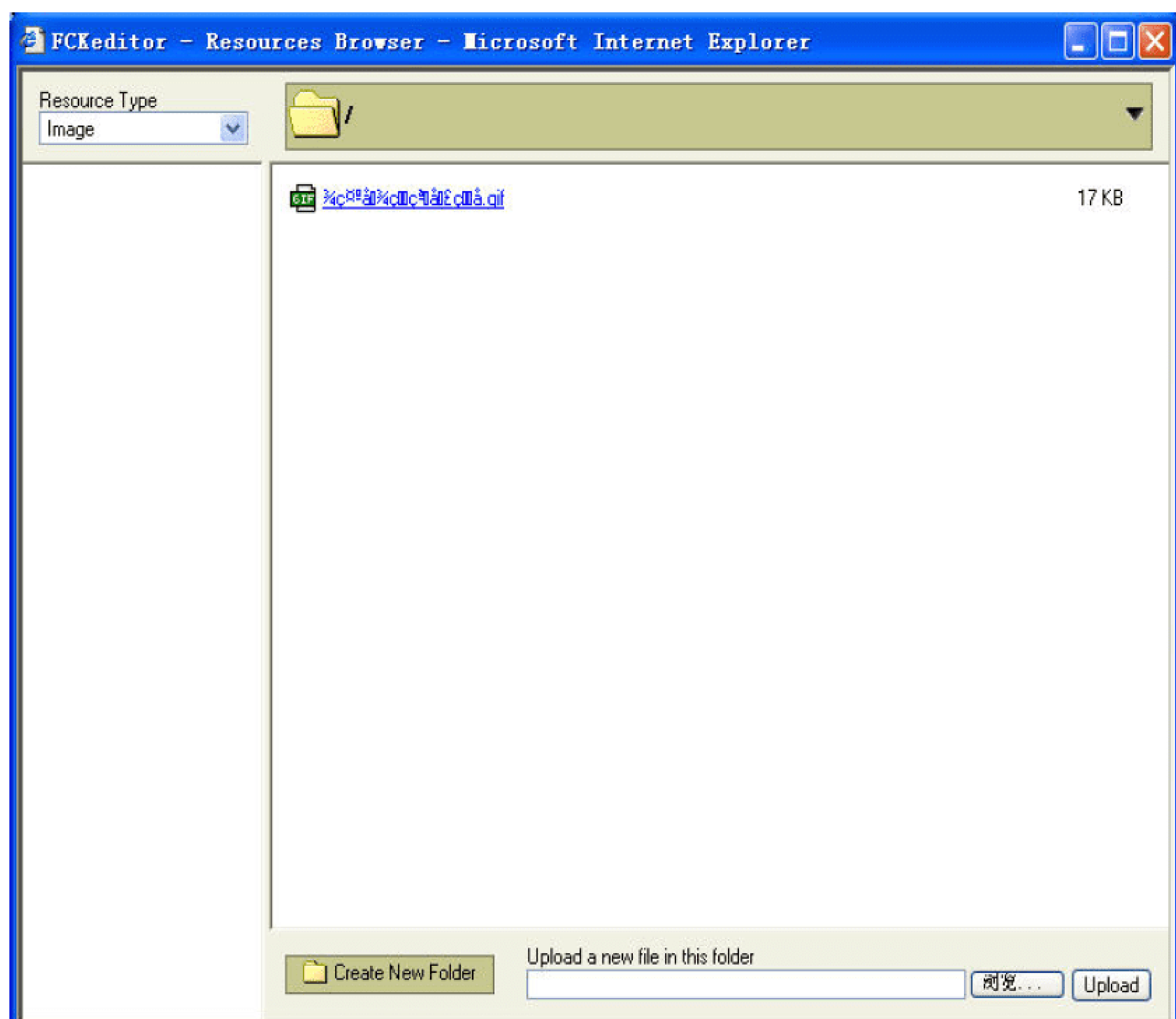


图 4.41　显示乱码名称的图像

1．页面编码情况

当文件上传时以 Post 方式发送请求，这时包含文件名的请求会以上传文件页面的编码方式进行编码。如果该页面的编码方式不是 utf-8，就会出现乱码形式。可以通过 frmupload.html 文件来查看上传文件页面的编码方式。

```
<html>
    <head>
        <title>File Upload</title>
        <!--编码方式 -->
```

```
        <meta http-equiv="Content-Type" content="text/html; charset=
        utf-8">
...
```

代码 charset=utf-8 证明不是上传文件页面的错误。

注意：为什么 frmupload.html 文件是上传文件页面？可以单击图 4.42 中的“浏览服务器”按钮，在出现的上传文件对话框中，单击右下方，在弹出的快捷菜单中选择“属性”选项就会弹出如图 4.43 所示的属性对话框。

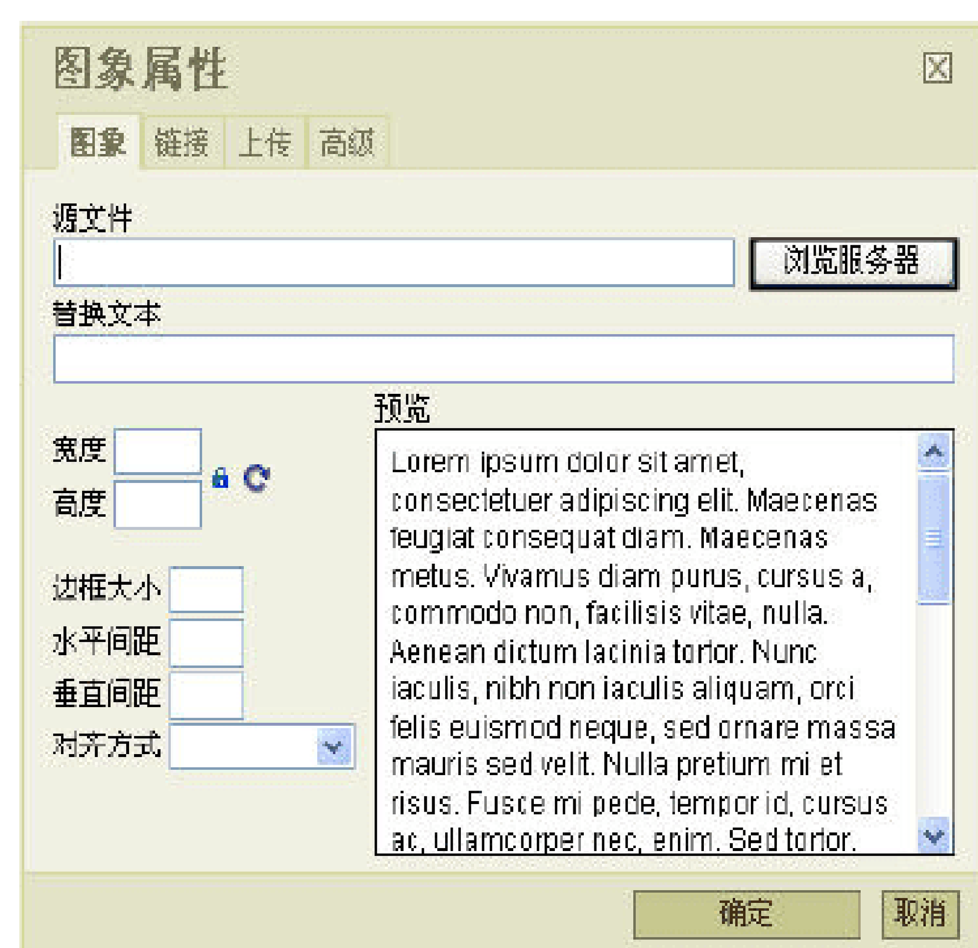


图 4.42　浏览服务器

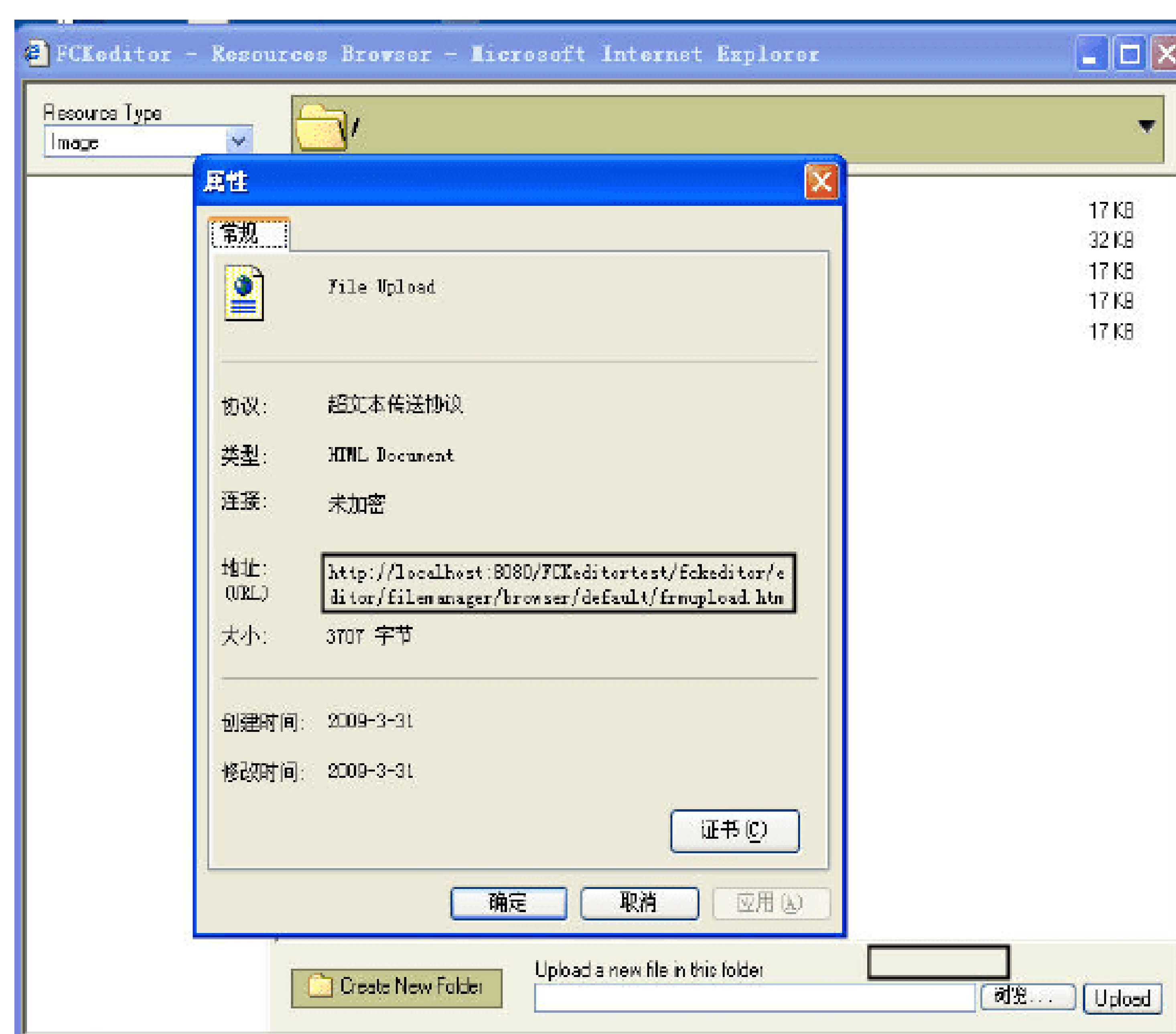


图 4.43　属性对话框

2．服务器的编码情况

当服务器接受请求后，在利用服务器端程序处理时如果没按照正确的编码方式处理，也会出现乱码。可以通过查看 web.xml 文件的配置信息找到服务器端程序 net.fckeditor.connector.ConnectorServlet，在该类的 doPost() 方法中可以发现应该在 List<FileItem> items = upload.parseRequest(request);这句代码前加入 upload.setHeaderEncoding("utf-8");这句代码来设置编码方式，如代码 4.11 所示。

代码 4.11　服务器端类：ConnectorServlet.java

```
public  void  doPost(HttpServletRequest  request,  HttpServletResponse
response)
        throws ServletException, IOException {
...
            try {
                upload.setHeaderEncoding("utf-8");     //编码方式
                List<FileItem> items = upload.parseRequest(request);
                FileItem uplFile = items.get(0);
...
      }
```

分析到出现乱码的原因后，就可以通过下面的步骤来解决。

（1）在 FCKeditortest/src 目录中新建一个名为 ConnectorServlet 的类，从 fckeditor-java-2.4.1-src.zip 源文件中找到 net.fckeditor.connector.ConnectorServlet 类并把其复制到新建

的 ConnectorServlet 的类中，同时要加入“upload.setHeaderEncoding("utf-8");”这句代码。

（2）接着修改 wel.xml 文件，代码 4.12 使得处理文件上传的类指向新建的 ConnectorServlet 类。

代码 4.12　配置上传文件类：web.xml

```
<!--配置 Connector 类路径-->
<servlet>
    <servlet-name>Connector</servlet-name>
    <servlet-class>ConnectorServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<!--配置 Connector 映射路径-->
<servlet-mapping>
    <servlet-name>Connector</servlet-name>
    <url-pattern>
        /fckeditor/editor/filemanager/connectors/*
    </url-pattern>
</servlet-mapping>
```

（3）单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮启动服务器，最后通过上传文件对话框上传名为“狗”的中文图像时就不会出现乱码，运行结果如图 4.44 所示。

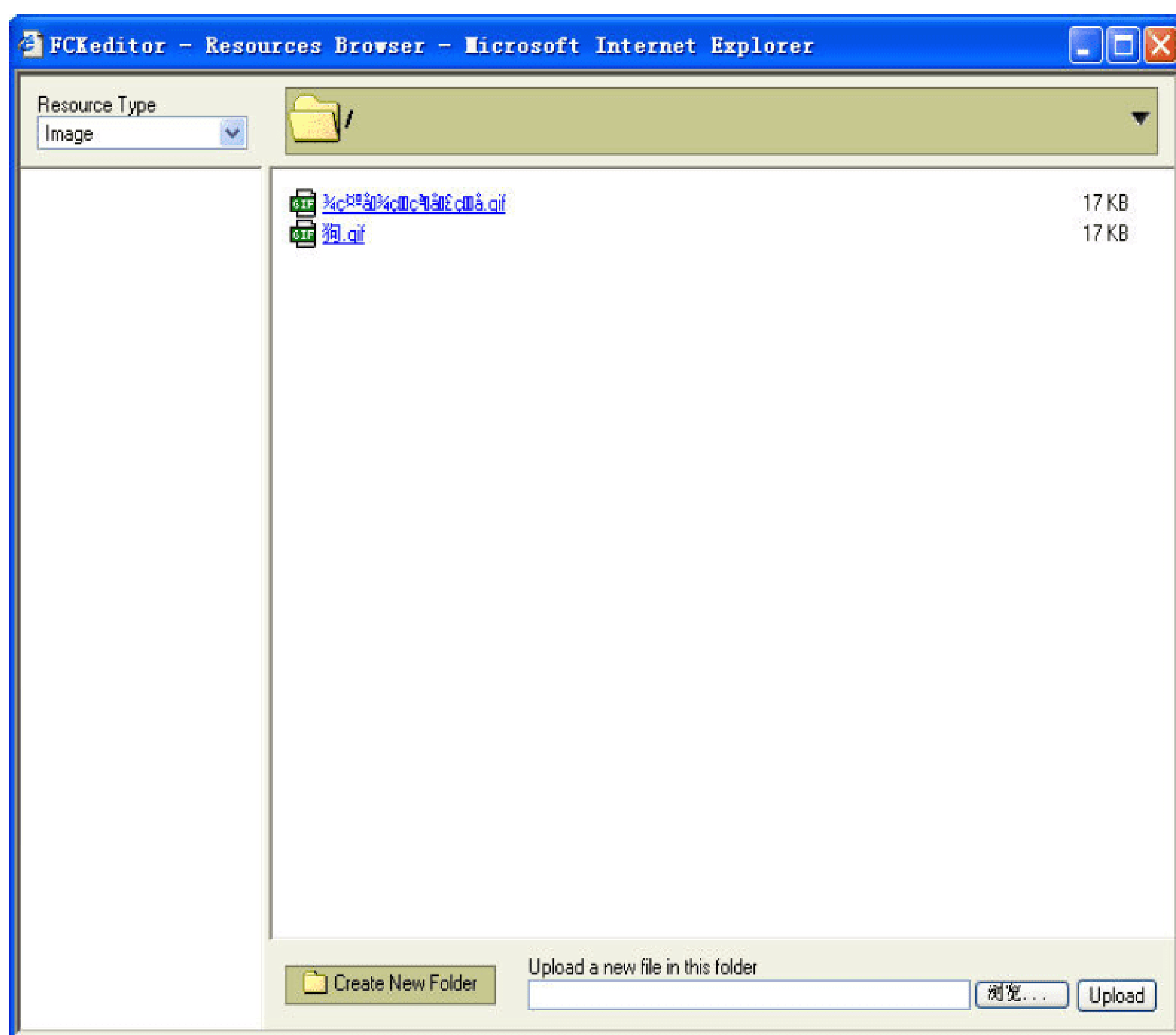


图 4.44　正确显示中文名图像

4.4.3　FCKeditor 在线文本编辑器上传文件配置——中文乱码（二）

经过 4.4.2 节的配置，虽然上传中文文件名的图像不会显示为乱码，但是当选择中文

名图像（狗.gif）来显示时就会出现如图 4.45 所示的显示错误。可以通过下面两种方案来解决来解决这种显示错误。

1．修改配置文件

Tomcat 服务器的配置文件为 Tomcat 6.0\conf\server.xml 文件，修改端口号为 8080 的连接器代码如下：

```
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443"
               URIEncoding="utf-8"
/>
```

🔔**注意：**当修改该文件时，必须先关闭 Tomcat 服务器。

当经过上述修改后，当服务器端处理中文名字时，就会转换成 URI 编码。重新启动服务器后，再通过上传文件对话框显示名为“狗.gif”的中文图像时就不会出现错误显示，如图 4.46 所示。



图 4.45　显示错误



图 4.46　正确显示中文名图像

2．修改Servlet类

为了避免保存中文名的图像，可以在服务器端类 ConnectorServlet 中，把中文转换成其他编码。在新建的 ConnectorServlet 类的 doPost()方法中找到保存文件的那段代码，然后修改内容如代码 4.13 所示。

代码 4.13　服务器端类：ConnectorServlet.java

```
...
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
...
                    //修改编码方式
                    filename=UUID.randomUUID().toString()+"."+extension;
                    if (!ExtensionsHandler.isAllowed(resourceType,
```

```
                   extension))
...
                   else {
                       File pathToSave = new File(currentDir, filename);
                       int counter = 1;
...
}
```

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮启动服务器，最后通过上传文件对话框上传名为“狗”的中文图像时就不会出现乱码，如图 4.47 所示。如果单击该图像的链接后，就会正确显示出该图像，如图 4.48 所示。

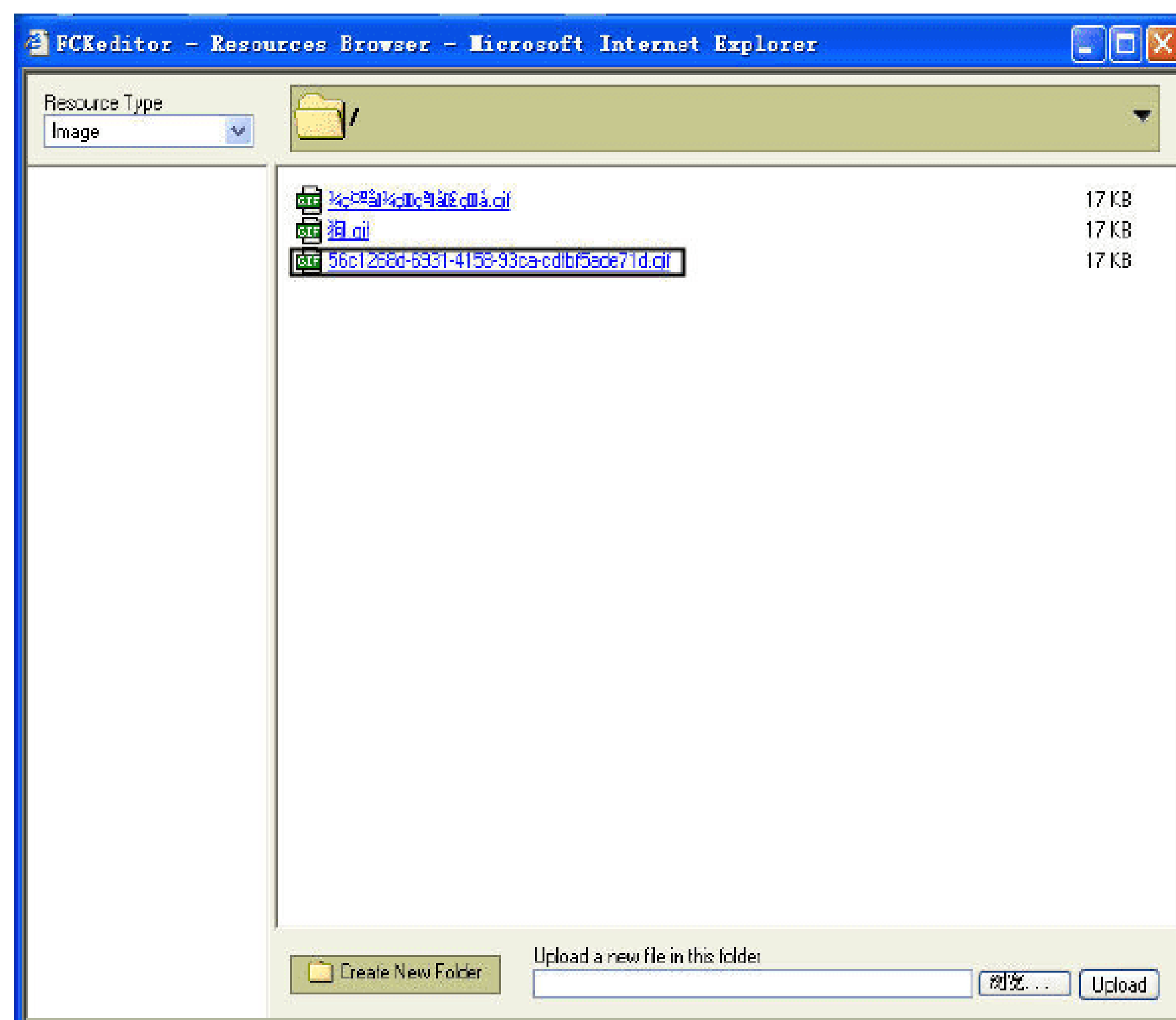


图 4.47　上传中文名图像



图 4.48　正确显示中文名图像

4.4.4　FCKeditor 在线文本编辑器配置上传文件类型

FCKeditor 在线文本编辑器允许上传文件（File）、图像（Image）、动画（Flash）和音频（Media），而每种类型都允许上传不同后缀名。例如，如果想通过上传文件对话框上传后缀名为 abc 的图像“狗.abc”时（如图 4.49 所示），就会出现如图 4.50 所示的错误。之所以会出现 Invalid file 错误，因为在 FCKeditor 在线文本编辑器中还没对 abc 后缀名的图像进行配置。

那么如何配置各个类型允许上传的后缀名？下面将通过修改配置文件使 FCKeditor 在线文本编辑器允许上传后缀名为 abc 的图像，具体步骤如下。

（1）首先配置服务器端的配置文件，即修改 FCKeditortest\src\fckeditor.properties 文件，为该文件添加如下代码。

```
connector.resourceType.image.extensions.allowed
=bmp|gif|jpeg|jpg|png|abc
```

（2）接着配置客户端的配置文件（config.js），在该文件中添加如下代码。

```
connector.resourceType.image.extensions.allowed
=bmp|gif|jpeg|jpg|png|abc
```

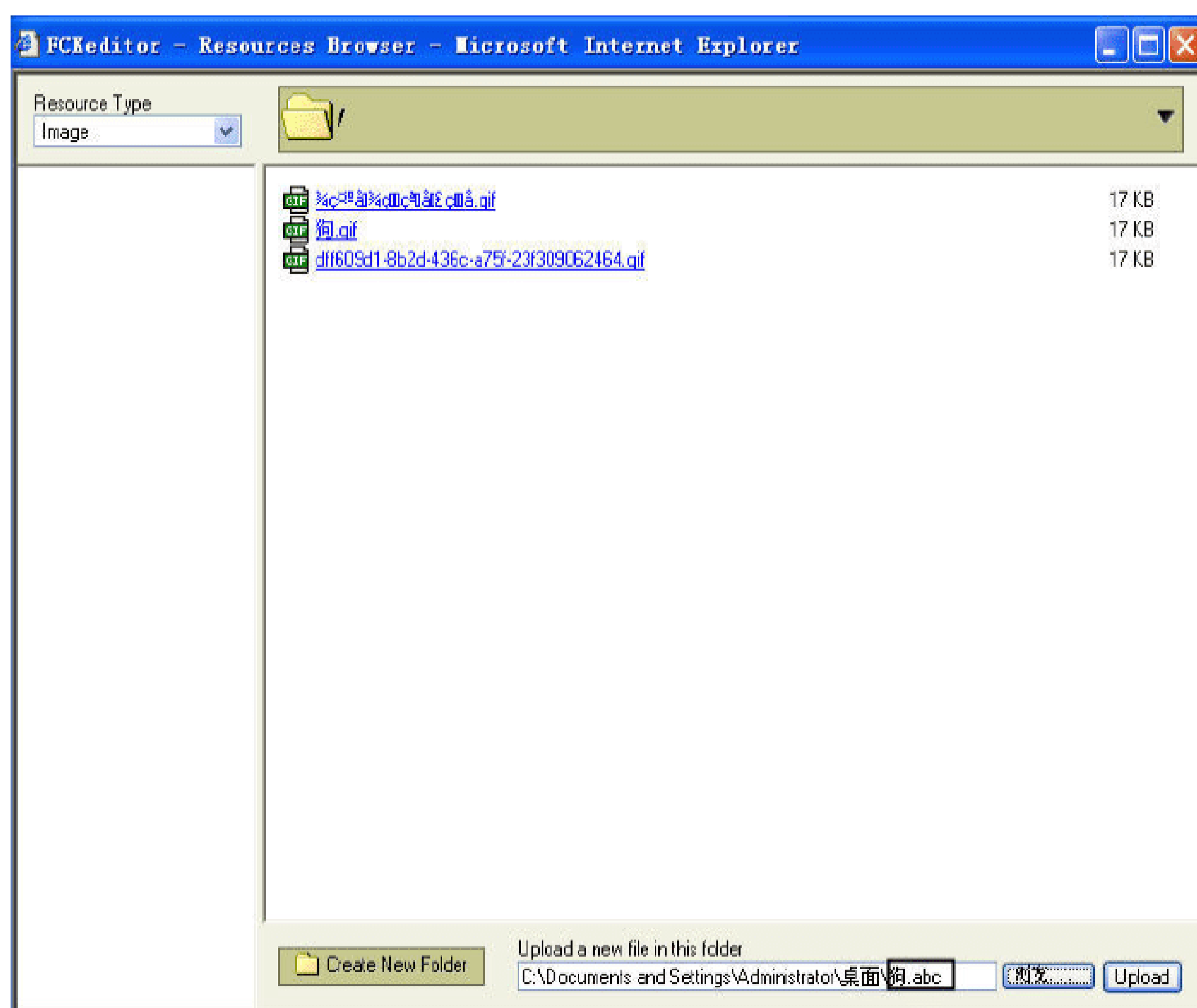


图 4.49　上传后缀名为 abc 的图像

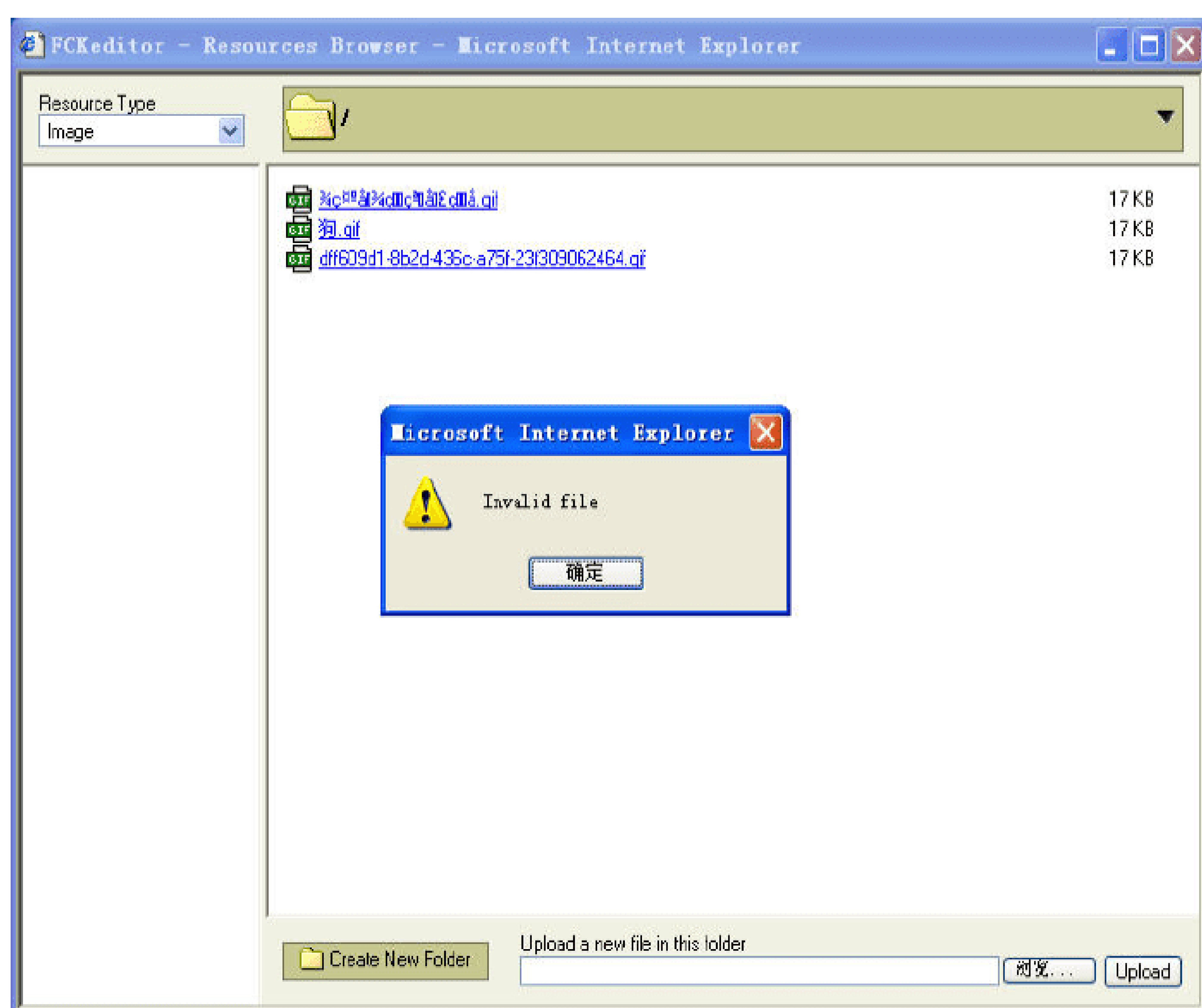


图 4.50　上传文件出错

注意：当配置允许上传的类型时，不仅要修改客户端的配置文件 config.js，同时还要修改服务器端的配置文件 fckeditor.properties。

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮启动服务器，最后通过上传文件对话框上传带有 abc 后缀名的“狗.abc”的图像时，就不会出现上传出错，如图 4.51 所示。同时如果单击该图像的链接，还会正确显示出该图像，如图 4.52 所示。

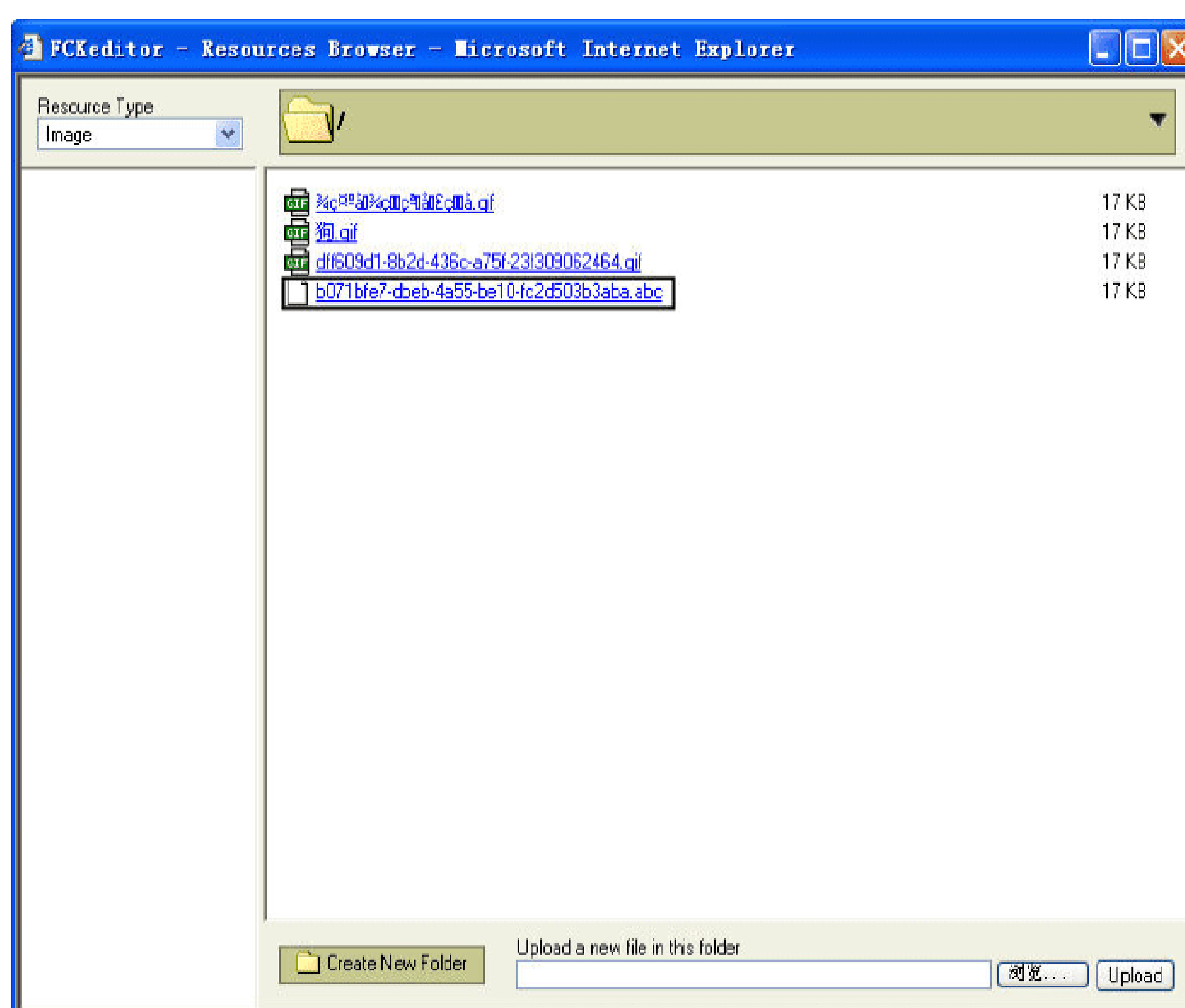


图 4.51　上传后缀名为 abc 的图像



图 4.52　显示后缀名为 abc 的图像

4.5　小　　结

本章详细讲解了在线编辑器 FCKeditor，在具体讲解该编辑器时，首先介绍如何下载 FCKeditor 组件，然后分析 FCKeditor 组件的相关目录和开发文档，最后分析了如何使用 FCKeditor 组件。

在具体介绍如何使用 FCKeditor 组件时，首先通过两个简单实例介绍如何使用 FCKeditor 组件，然后详细介绍了 FCKeditor 组件的相关配置选项，最后介绍了 FCKeditor 组件的高级应用：上传文件。

第 5 章　验证模块（JSP+Servlet+JSValidation）

验证模块对于任何网络系统来说是一个常见而不可缺少的模块，对于每一个网络系统的前台页面程序来说几乎是必需模块。由于验证模块总是离不开表单，所以编写一个功能完善的表单是实现验证模块的基础。

本章除了详细介绍如何实现验证模块外，还介绍表单的各个方面：如何通过 JSValidation 框架实现表单验证、如何对输入内容进行验证、如何避免表单重复提交和如何实现图像的缩略加水印功能。

5.1　表 单 基 础

表单最基本的应用就是收集信息和反馈意见，复杂应用有资料检索、讨论、网上购物等多种交互式操作。表单的这种信息交互式特点，使得其不再是一个单一的信息发布载体，而是根据客户提交的信息动态地进行信息重组。

5.1.1　表单的基础内容

当创建一个表单页面时，经常会包含一些基础内容。这些内容包括单行文本框、密码框、多行文本框、单选按钮、复选框和下拉菜单，其显示页面如图 5.1 所示。各个元素的实现如下：

姓名：
密码：
年纪：○小于18 ○18 - 25 ○26-40 ○大于 40
编程时间：不编程
使用的操作系统：Win XP / Win 2000/2003 / Linux / FreeBSD / Mac OS / other
使用的编程语言 □C □C++ □C# □PYTHON □JAVA □VB □DEPHI
建议：
reset　submit

图 5.1　表单页面

（1）在上述表单页面中，表单元素使用下面代码表示。

```
<form method="POST" action="" >
```

元素<form>用来创建表单，属性 method 用来设置该表单的提交方式，属性 action 的值为动态处理该表单提交参数的程序路径。

（2）在上述表单页面中，“姓名”后面的文本框使用下面代码表示。

```
<input type="text" name="name" size="40">
```

type="text"用来创建单行文本框，属性 name 为单行文本框的名字，属性 size 用来设置文本框的宽度。

注意：size 设置文本框的长度，而属性 maxlength 设置允许输入的最长字符数。

（3）在上述表单页面中，“密码”后面的密码框使用下面代码表示。

```
<input type="password" name="password" size="40">
```

type="password"用来创建密码框，属性 name 为密码框的名字，属性 size 用来设置密码框的宽度。

注意：虽然输入的密码将以*显示在密码框中，但是密码并没有加密，只是被*替换显示。

（4）在上述表单页面中，“年纪”后面的单选按钮使用下面代码表示。

```
<input type="radio" name="age" value="18" checked>小于 18
<input type="radio" name="age" value="18-25">18 - 25
...
```

type="radio"用来创建单选按钮，属性 name 为单选按钮的名字，属性 value 用来设置单选按钮的值，属性 checked 表示默认的选项。

注意：如果属性 name 值相同，则表示这些单选按钮属于同一组。

（5）在上述表单页面中，“编程时间”后面的下拉列表框使用下面代码表示。

```
<select name="codetime" size="1">
 <option value="never">不编程
...
</select>
```

<select>元素用来创建下拉菜单，属性 size 用来设置显示的选项数目。子元素<option>用来创建选项，其属性 value 用来设置该选项的值。

（6）在上述表单页面中，“使用的操作系统”后面的下拉菜单使用下面代码表示。

```
<select name="os" size="6" multiple>
 <option value="WinXP">Win XP</option>
 ...
</select>
```

该下拉列表与上面名为 codetime 的下拉列表的区别，在于元素<select>中设置了 multiple 属性，即可以通过使用 Ctrl 和 Shift 键进行多选。

（7）在上述表单页面中，“使用的编程语言”后面的复选框使用下面代码表示。

```
<input type="checkbox" name="language" value="C++">C++
<input type="checkbox" name="language" value="C#">C#
...
```

type="checkbox"用来创建复选框，属性 name 为复选框的名字，属性 value 用来设置复选框的值。

（8）在上述表单页面中，“建议”后面的多行文本框使用下面代码表示。

```
<textarea name="comment" cols="40" rows="4"></textarea>
```

<textarea>用来创建多行文本框，属性 cols 用来设置显示时可以容纳的字符列数宽度。属性 rows 用来设置可以容纳的字符行数高度。

（9）在上述表单页面中，最后一行的两个按钮使用下面代码表示。

```
<input type="reset" value="reset">
<input type="submit" value="submit">
```

type="submit"用来创建 Submit 按钮，实现表单的提交。type="reset"用来创建 Reset 按钮，用来恢复常态。

注意： 表单中还有一个非常重要元素被称为隐藏元素，其不会在页面界面上显示，但是在需要跨页之间传值时，可以使用该元素传递一些隐含的值，具体代码如下：

```
<input type="hidden" name="xxx" value="xxx">
```

5.1.2　表单必备功能

在本节中，以直观的方式向读者介绍在开发具体的表单时需要实现的一些必备功能。这些功能包括表单验证、避免重复提交和在线文本编辑器。

首先介绍一下 form 项目，在该项目中实现了通过 JSValidation 框架实现表单验证、实现对输入内容进行验证、利用客户端 JavaScript 语言避免表单重复提交，具体目录如图 5.2 所示。

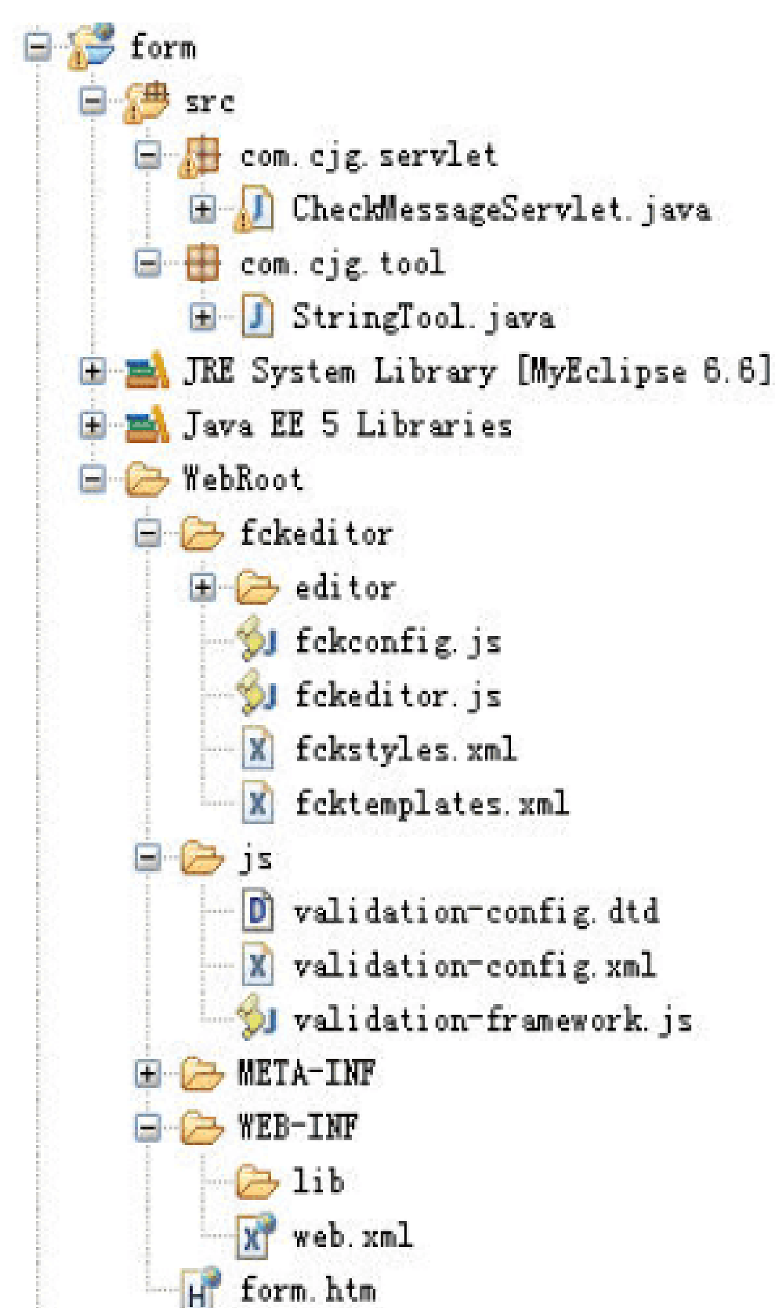


图 5.2　目录结构

（1）在上述表单页面 form.htm 中，通过 JSValidation 框架实现对一些内容的验证。例如当“姓名”内容为空，单击“提交”按钮就会出现如图 5.3 所示的页面；当 E-mail 内容格式错误，单击“提交”按钮就会出现如图 5.4 所示的页面；当“主题”内容为空，单击“提交”按钮就会出现如图 5.5 所示的页面。

（2）在上述表单页面 form.htm 中，通过 JavaScript 语言避免表单的重复提交。即当相应的选项填写了正确的内容后，连续单击“提交”按钮后，就会出现如图 5.6 所示的页面。

（3）在上述表单页面 form.htm 中，“内容”选项是 FCKeditor 在线文本编辑器（如图 5.7 所示），使用该编辑器可以对填写的内容进行格式的设置，同时还可以实现上传文件等功能。

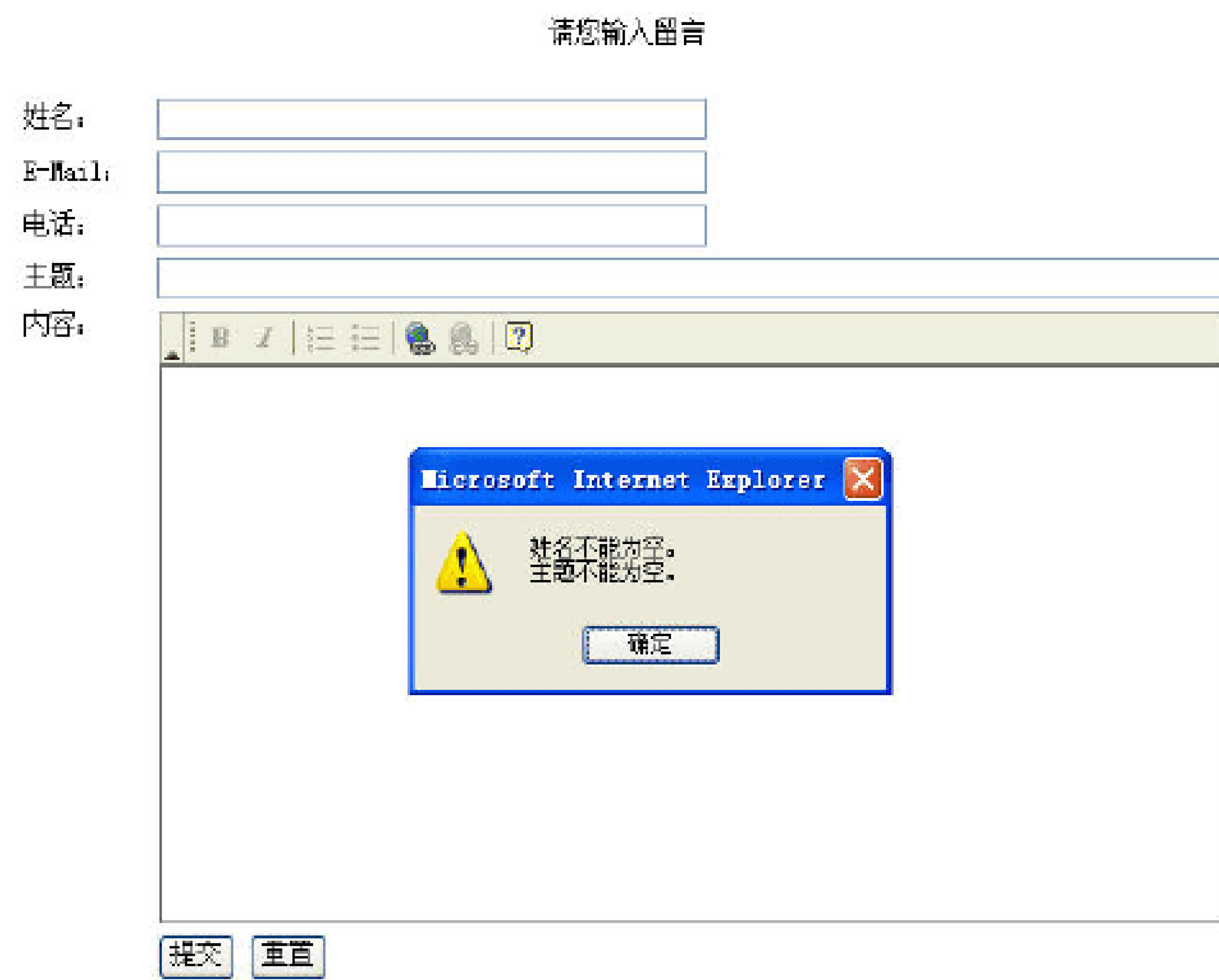


图 5.3　对姓名验证

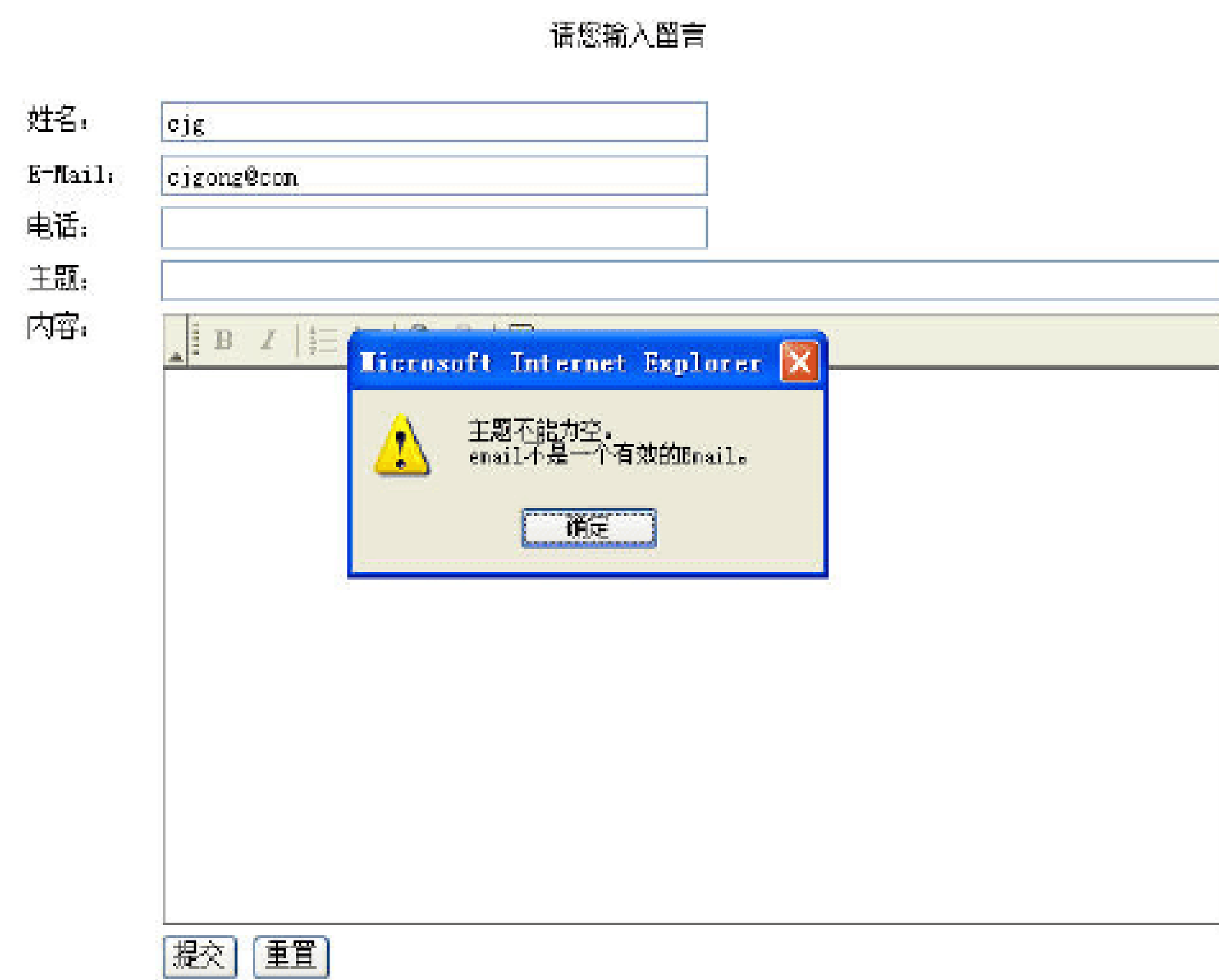


图 5.4　对 E-mail 格式验证

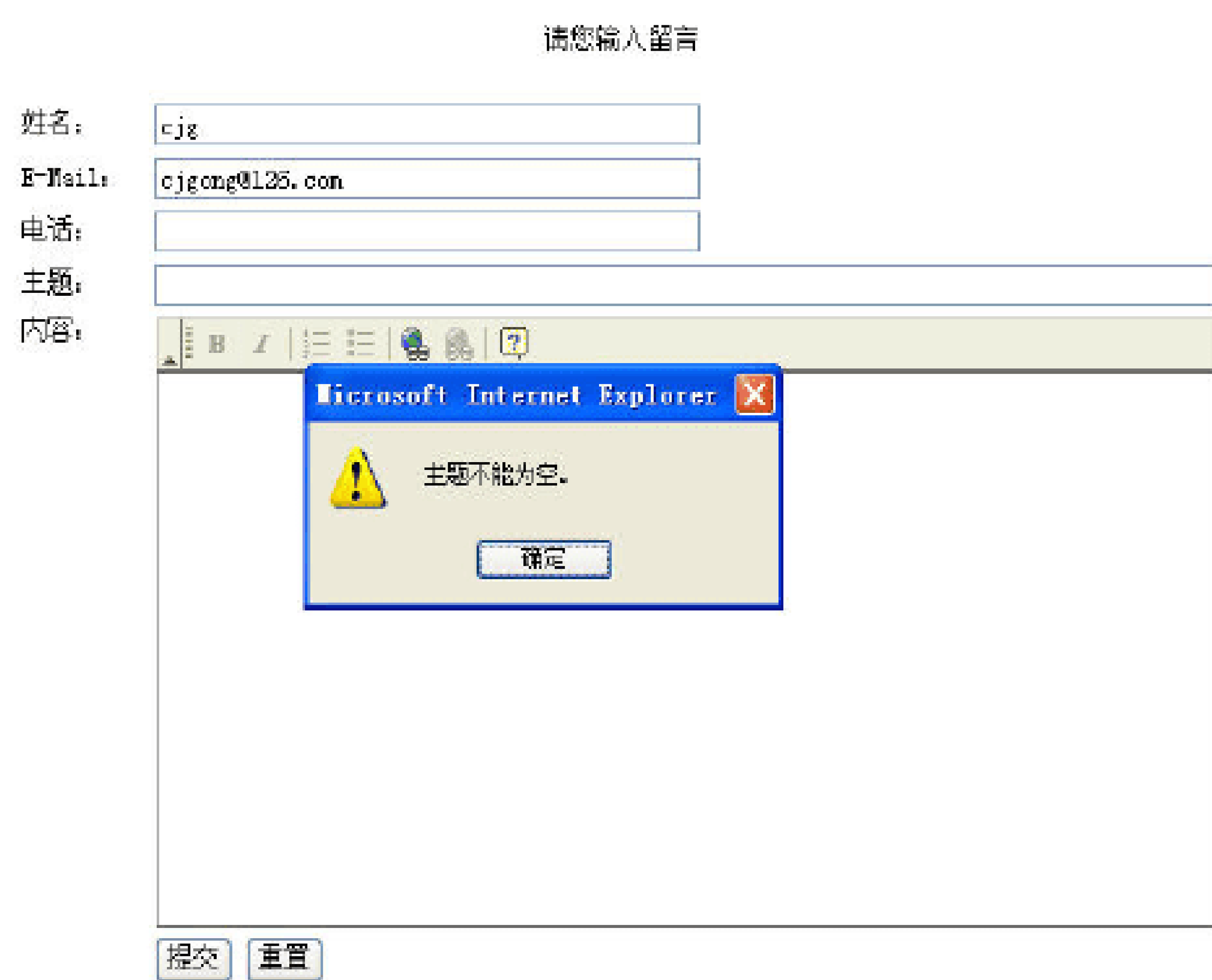


图 5.5　对主题验证



图 5.6　避免重复提交

（4）单击工具栏上的![]按钮，把该项目发布到服务器。然后单击工具栏上的![]按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/form/form.htm，运行结果如图 5.8 所示，输入相应内容后，单击“提交”按钮后，弹出如图 5.9 所示的显示内容的页面。

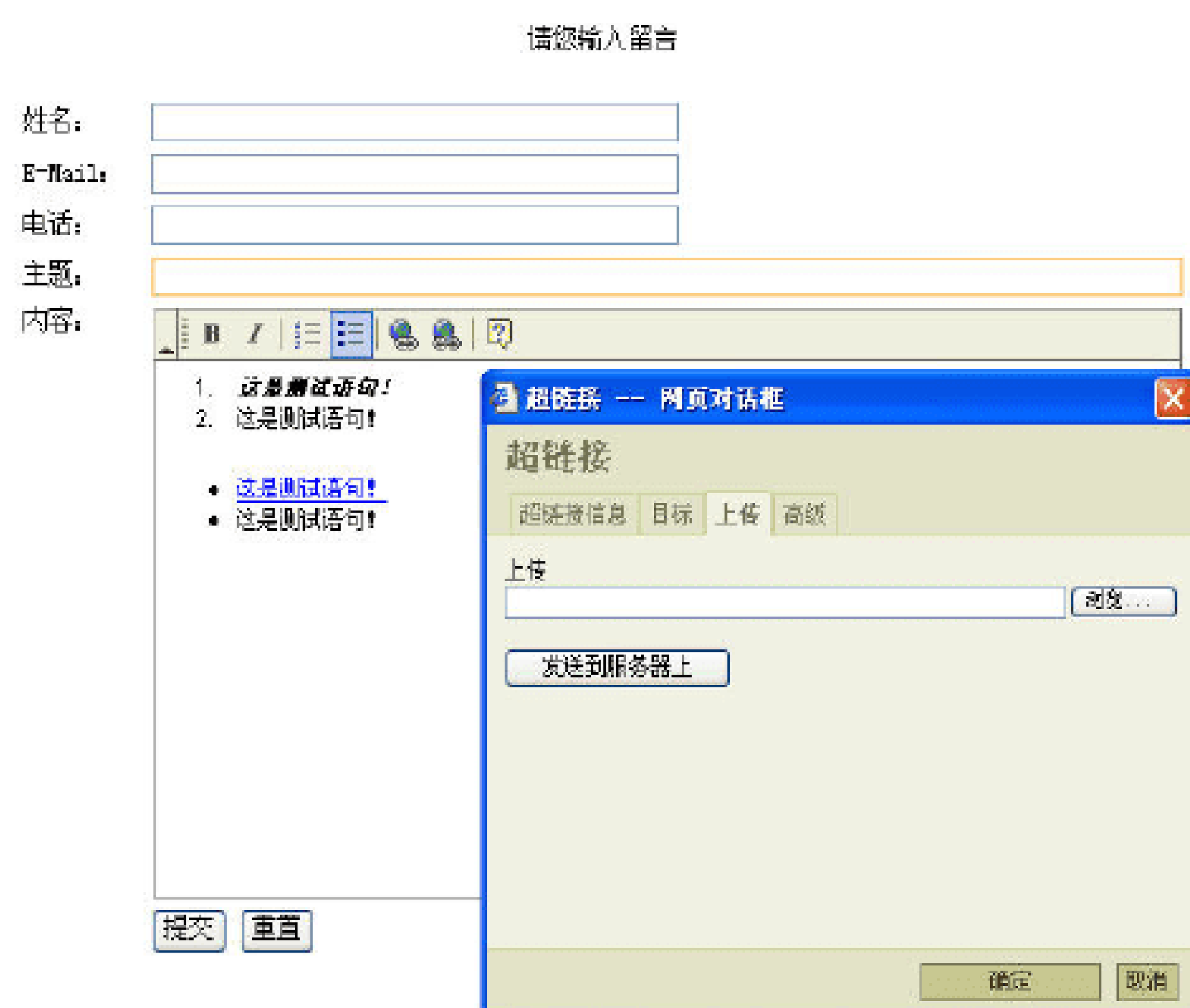


图 5.7　FCKeditor 在线文本编辑器

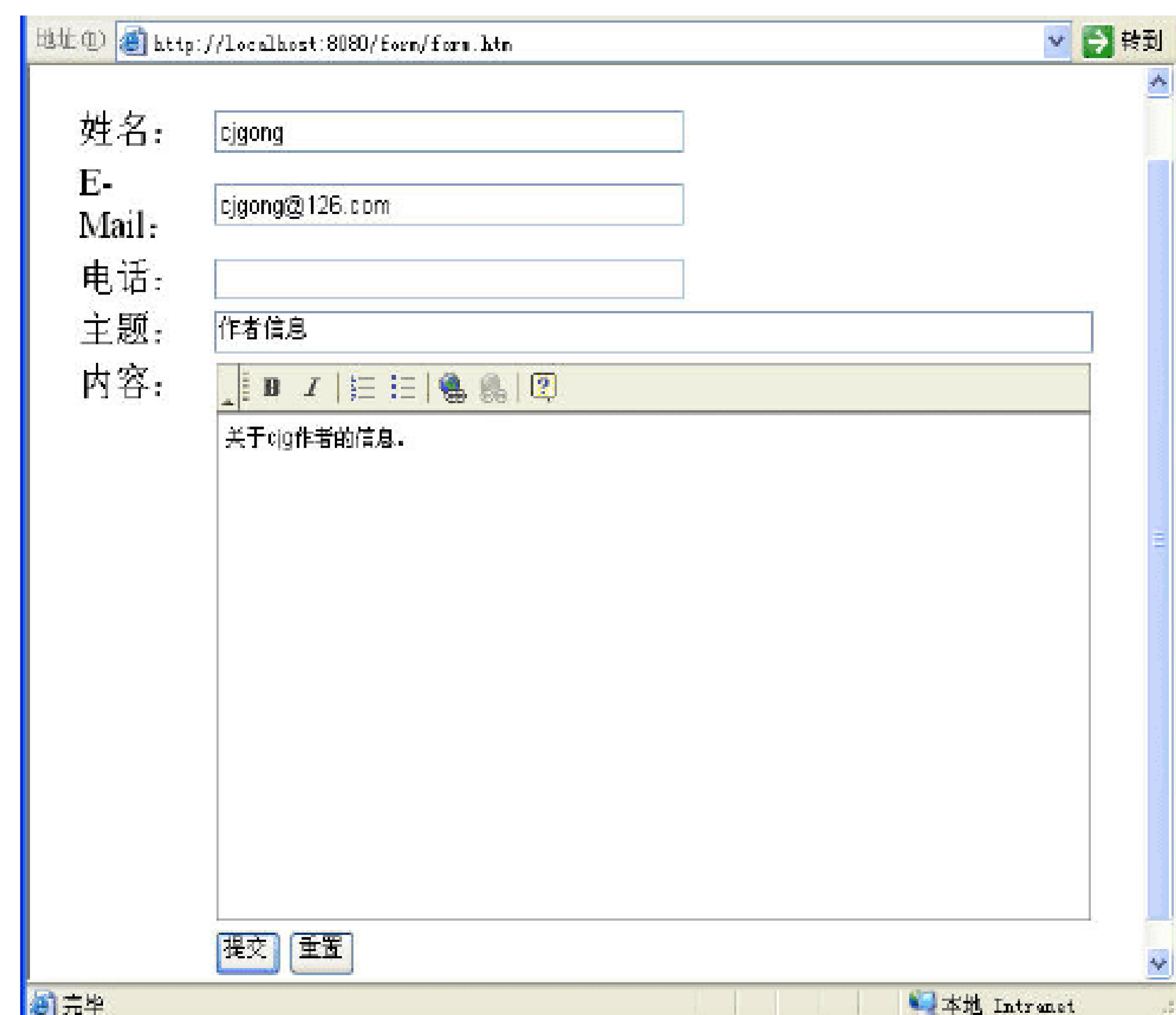


图 5.8　添写内容

接着介绍一下 verificationcode 项目，在该项目中通过服务器端程序实现验证码，具体目录结构如图 5.10 所示。



图 5.9　显示内容

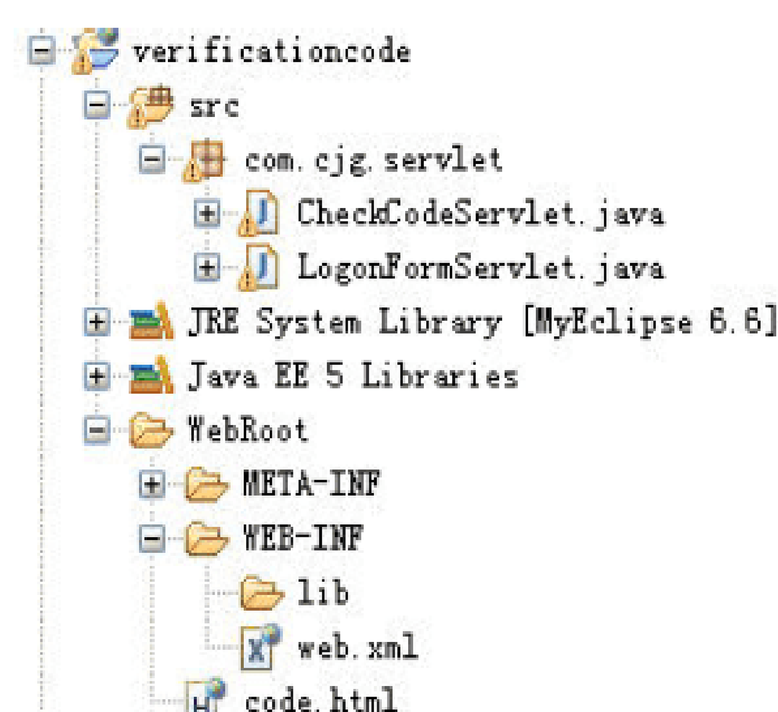


图 5.10　目录结构

发布该项目后，通过浏览 http://localhost:8080/verificationcode/code.html 地址就会显示出登录界面，填写相应的信息后（如图 5.11 所示），只要单击“登录”按钮就会显示出登录正确的信息，如图 5.12 所示。如果在图 5.11 中的验证码填写错误，就会转到如图 5.13 所示的验证码错误页面。



图 5.11　登录页面

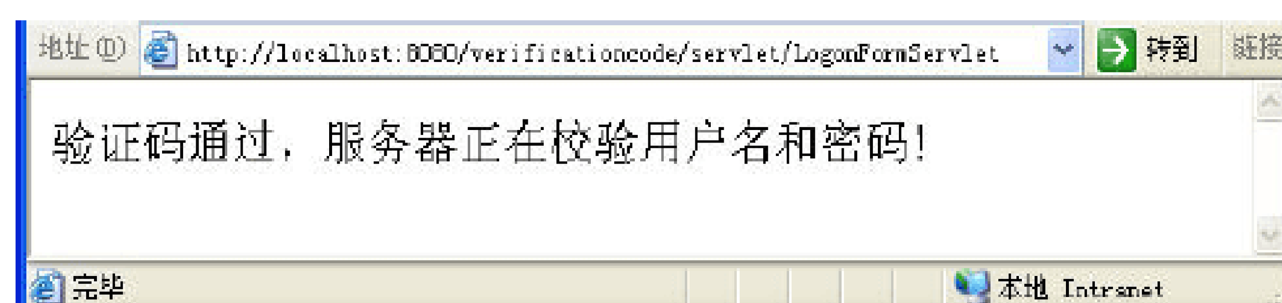


图 5.12　正确登录



图 5.13　验证码错误页面

5.1.3　表单功能具体实现

form.htm 页面是用来实现表单页面的代码。该页面主要用于收集信息，即需要浏览者在相应的选项填写相应的内容。代码 5.1 用来实现信息的收集。

代码 5.1　收集信息：form.html

```
...
<style>                                                     <!--样式表-->
{
    font-family: "宋体";
    font-size: 15px
}
</style>
<!--引入 validation-framework.js 文件和 fckeditor.js -->
```

```
<script type="text/javascript" src="${ctx}/js/validation-framework.js">
</script>
<script type="text/javascript" src="${ctx}/fckeditor/fckeditor.js">
</script>
<!--设置避免表单重复提交功能-->
<script language="text/javascript">
             var checkSubmitFlg=true;          <!--设置一个标记变量-->
             function checkSubmit()            <!--设置 checkSubmit()-->
             {
                 if(true==checkSubmitFlg)      <!--判断标记变量-->
                 {
                     document.form1.submit();  <!--提交表单-->
                     checkSubmitFlg=false;     <!--设置一个标记变量-->
                 }
                 else
                 {
                     alert("你已经提交了表单，请不要重复提交！");
                 }
             }
</script>
<p align="center">
    请您输入留言
</p>

<form id="form1" name="form1" method="post" action="/form/servlet/
CheckMessage"
    onsubmit="return doValidate(this)">
...
                <!--各种文本框-->
                姓名:
                <input name="name" type="text" id="name" size="50"
                maxlength="20" />
                E-mail:
                <input name="email" type="text" id="email" size="50"
                maxlength="50" />
                电话:
                <input name="phone" type="text" id="phone" size="50"
                maxlength="20" />
                主题:
                <input name="title" type="text" id="title" size="80"
                maxlength="80" />
                内容:
                <!--关于 FCKeditor 编辑器-->
                <script type="text/javascript">
                var oFCKeditor = new FCKeditor("content");
                oFCKeditor.BasePath= '${ctx}/fckeditor/' ;
                oFCKeditor.Height  = 300 ;
                oFCKeditor.ToolbarSet = 'Basic';
                oFCKeditor.Create() ;
                </script>
                <!--表单的相关按钮 -->
                <input type="submit" name="Submit" value="提交"
                    onClick="checkSubmit();" />
                <input type="reset" name="Reset" value="重置" />
...
</form>
...
```

【代码解析】

- 在上述代码中，文件头和文件尾都是通过引入外部文件来实现的。在 form.html 页面中不仅通过 JSValidation 框架实现内容的验证，而且还实现了 FCKeditor 在线文本编辑器。
- 当配置 FCKeditor 在线文本编辑器的工具栏时，通过如下代码实现基础工具栏。

```
oFCKeditor.ToolbarSet = 'Basic';
```

- 对于客户端表单验证框架（JSValidation），首先修改 formtest/WebRoot/js/validation-framework.js 文件中的第一句内容如下：

```
var ValidationRoot = "/formtest/js/";
```

接着编写 validation-config.xml 文件的内容，用来实现对姓名、主题和 E-mail 这 3 个选项进行验证，具体内容如代码 5.2 所示。

代码 5.2　客户端验证：validation-config.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE validation-config SYSTEM "validation-config.dtd">
<validation-config lang="auto">
    <form id="form1" show-error="alert" show-type="all">
        <!--针对姓名不为空进行验证-->
        <field name="name" display-name="姓名" onfail="">
            <depend name="required" />
            <depend name="minLength" param0="2" />
            <depend name="maxLength" param0="20" />
        </field>
        <!--针对主题不为空进行验证-->
        <field name="title" display-name="主题">
            <depend name="required" />
        </field>
        <!--针对 E-mail 格式进行验证-->
        <field name="email" display-name="email">
            <depend name="email" />
        </field>
    </form>
</validation-config>
```

注意： 虽然通过 JavaScript 语言在客户端可以避免表单的重复提交，但是在具体使用时却存在一定的缺陷。

5.2　客户端表单验证框架

form 项目中利用客户端表单验证框架（JSValidation），分别实现了对姓名和主题不为空的验证和对 E-mail 格式的验证。本节将详细介绍如何下载客户端表单验证框架、如何配置客户端表单验证框架，以及如何使用客户端表单验证框架。

5.2.1　下载客户端表单验证框架（JSValidation）

JSValidation 全称为客户端表单验证框架，其是采用 XML 和 JavaScript 相结合的方式进行客户端校验的一种表单验证框架。目前最稳定的版本为 1.0b5 版本，具体的下载步骤如下。

（1）访问下载 JSValidation 框架的官方网站（http://cosoft.org.cn/），如图 5.14 所示。



图 5.14　JSValidation 框架下载首页

（2）打开 JSValidation 框架官方网站的首页后，在其左边的项目搜索文本框中输入 JSValidation，单击 Go 按钮就会转到如图 5.15 所示的搜索结果页面。

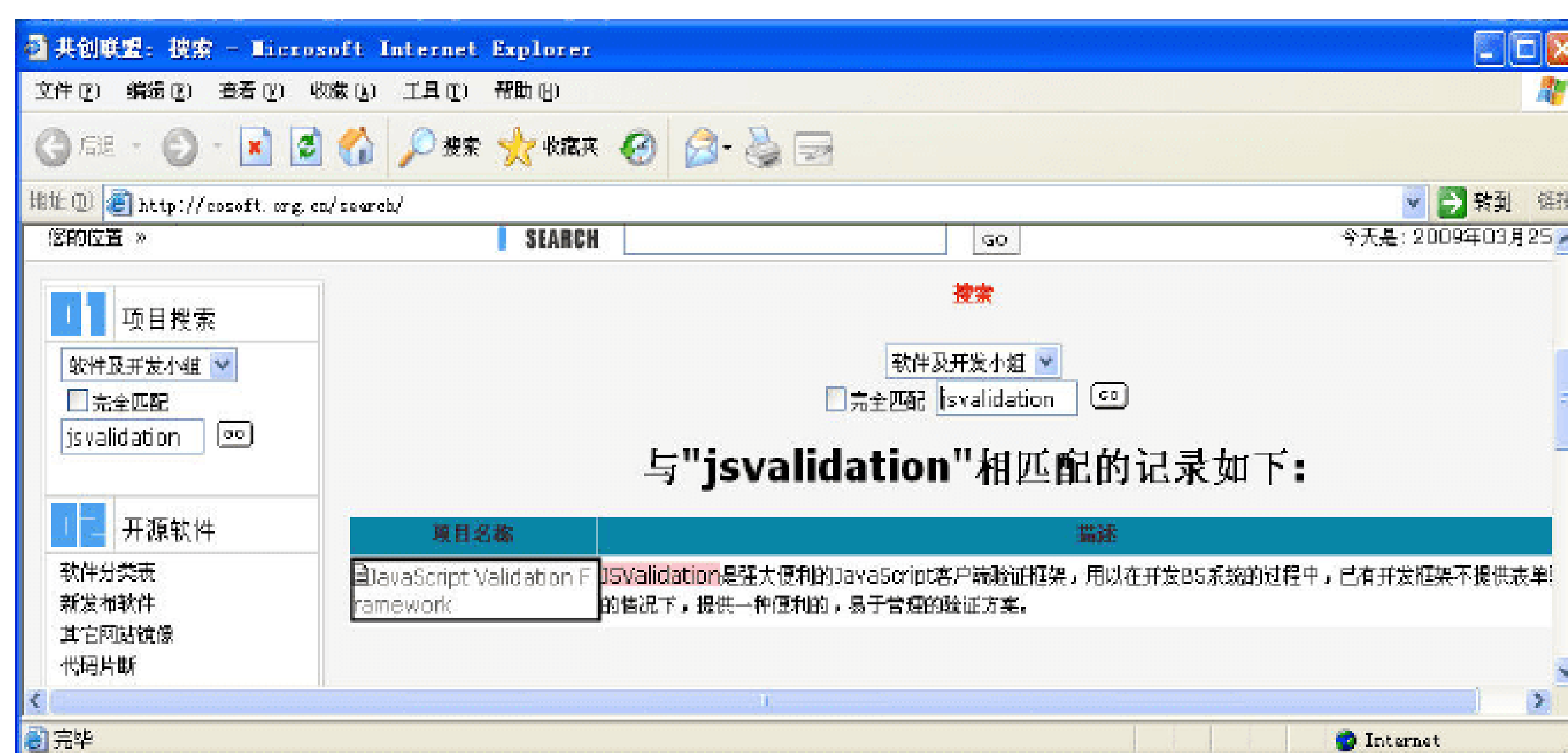


图 5.15　搜索结果页面

（3）在搜索结果页面中，单击 JavaScript Validation Framework 链接后，就可以转到如图 5.16 所示的下载页面。

（4）单击“下载”链接后，就可以进入真正下载页面（如图 5.17 所示）。单击 1.0b4 目录下的 jsvalidation-1_ob4.zip 链接后，就会实现下载该框架。

至此，就完成对 JSValidation 框架的下载。

5.2.2　JSValidation 表单验证框架使用

5.2.1 节介绍了如何下载 JSValidation 框架，下载完该框架后就可以在 Java Web 项目中

使用该框架。在具体使用之前，先解压 jsvalidation-1_0b4.zip 文件，该压缩包目录如图 5.18 所示。

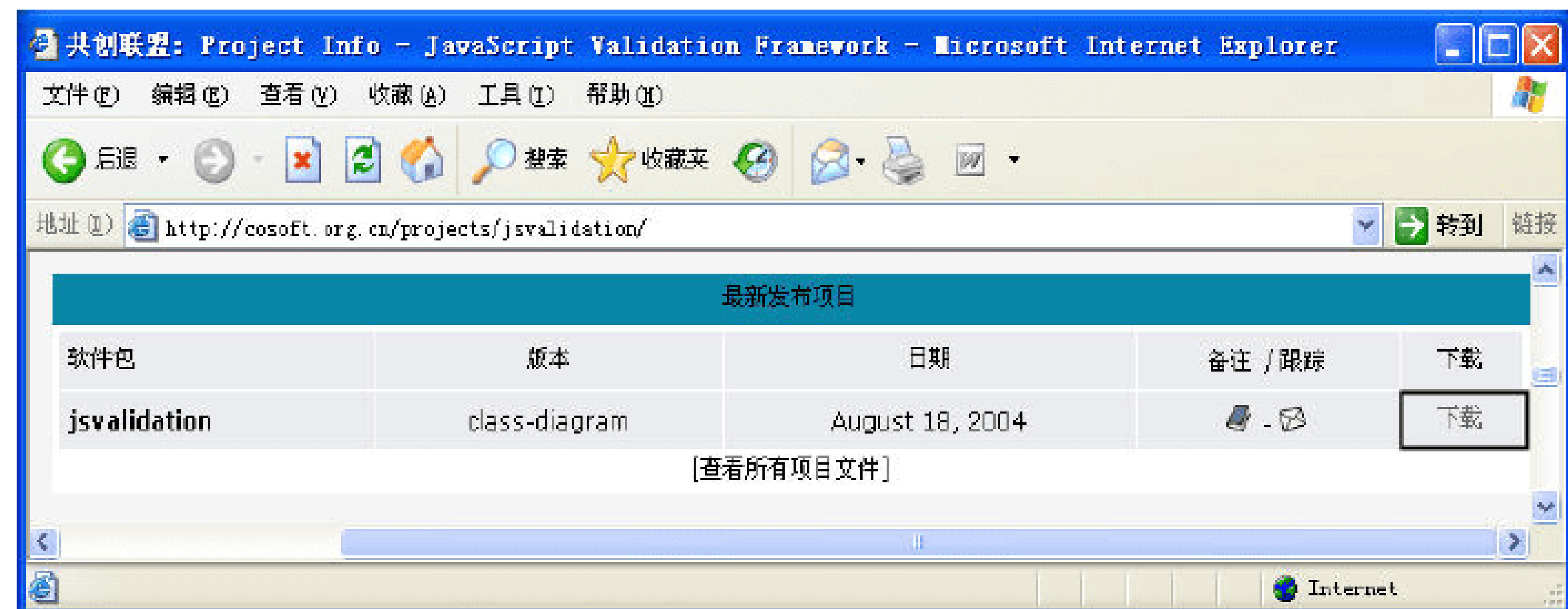


图 5.16　下载页面

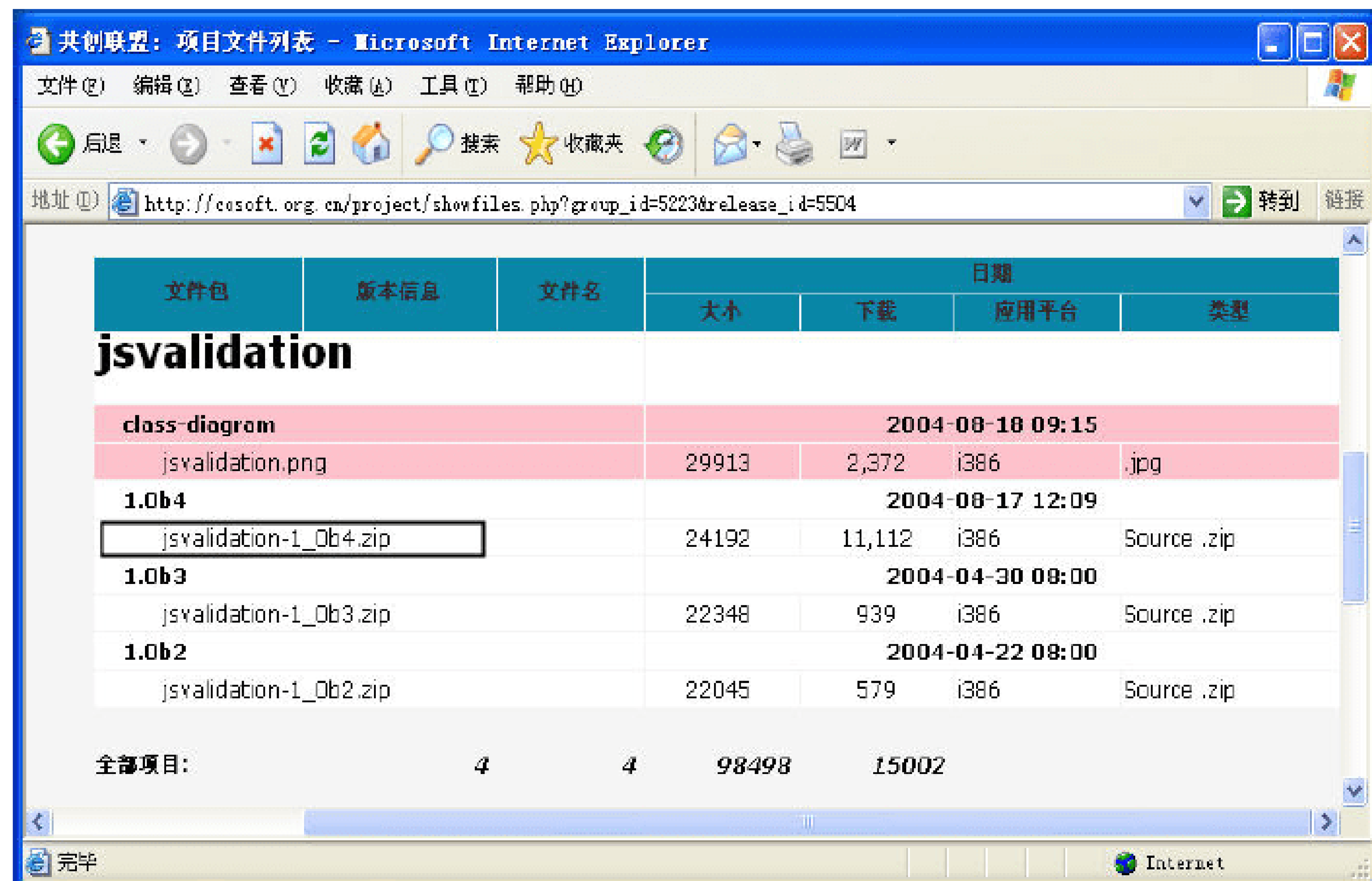


图 5.17　下载 1.0b4 版本

名称	大小
..	
whatsnew.txt	1,523
validation-framework.js	32,150
validation-config.xml	1,421
validation-config.dtd	603
userguide.html	13,940
todo.txt	357
style.css	1,531
site.js	913
index.html	4,776
faq.html	2,703
download.html	1,330
devguide.html	1,052
demo_result.html	1,193
demo.html	2,823
.project	216

图 5.18　目录结构

上述目录中的 3 个文件构成了 JSValidation 框架的主体结构，其他都是 Demo。这 3 个文件分别如下。

- validation-config.xml：用来编写相应校验规则的配置文件。
- validation-config.dtd：用来定义 validation-config.xml 的文档结构。
- validation-framework.js：用来解析 validation-config.xml 和实现校验功能的函数

文件。

下面将具体介绍如何在 Java Web 项目中使用 JSValidation 框架，具体步骤如下。

（1）首先应该把 jsvalidation-framework.、jsvalidation-config.dtd 和 validation-config.xml 复制到 fckeditortest 网站 WebRoot 目录下的 javascript 文件里（目录如图 5.19 所示），以达到配置环境的目的。

（2）修改配置文件。首先打开 jsvalidation-framework.js 文件，修改第一行代码如下：

```
var ValidationRoot = "/JSValidationtest/javascript/";
```

该变量的值为 validation-config.xml 文件的位置。接着修改 validation-config.xml 文件，该文件用来集中管理表单的存放点，也是 JSValidation 框架处理验证条件的地方。其文档结构如图 5.20 所示。每个节点的属性如表 5.1 所示，各个节点的作用如下。

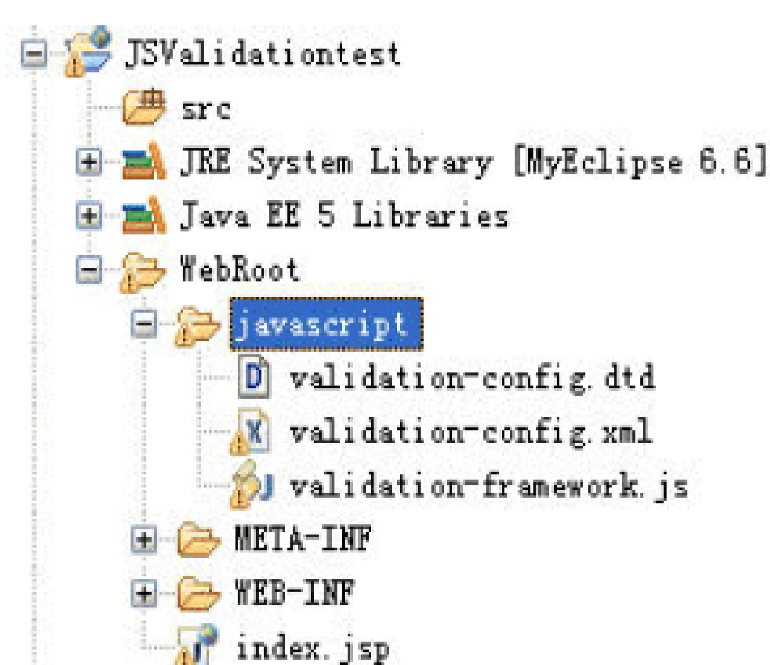


图 5.19　目录结构



图 5.20　文档结构

- <validation-config>：其为 validation-config.xml 文件的根节点。
- <form>：虚拟表单元素，其可以设置多个，用来进行一个或者多个 form 的校验。
- <field>：虚拟表单域元素，其也可以设置多个，表示 form 中有一个或者多个需要验证的表单域。
- <depend>：检验条件元素，其也可以设置多个，表示每个域需要验证的条件可以有一个或者多个。

表 5.1　节点属性

节　　点	属　　性	意　　义
validation-config	lang	其值可以为“auto”、“zh-cn”（中文）和“en-us”（英文），默认为 auto，用来设置所用的语言
form	id	用来与实际网页表单相关联
form	show-error	用来设置错误提示信息的方式。其值可以是 alert 或某个 div 的 id
form	onfaile	用来设置当检验失败后，需要运行的 JavaScript 的函数
field	name	用来与实际网页表单中的域相惯关联
field	Display-name	用来设置当检验失败后，会显示的名称
field	onfaile	用来设置当检验失败后，需要运行的 JavaScript 的函数
depend	name	用来设置检验条件的名称
depend	param	用来设置检验条件的参数

（3）修改需要进行检验的页面，在该页面中首先加入 JSValidation 框架的引用。

```
<script language="javascript" src="/JSValidationtest/javascript/
```

```
jsvalidation-framework.js"></script>
```

接着在需要验证的表单 Form 标记中，加入如下代码：

```
onsubmit="return doValidate('formId')"
```

其中参数 formId 是该表单 Form 自己的 ID，代码 5.3 为用来实现验证的测试表单。

代码 5.3　测试表单：test.html

```
...
    <body>
        <!--引入 validation-framework.js 脚本文件-->
        <SCRIPT LANGUAGE="JavaScript"
            src="/JSValidationtest/javascript/validation-framework.js">
            </SCRIPT>
...
                        <!--对 onsubmit 进行配置-->
                        <form method="GET" action="demo result.html"
                        id="form1"
                            onsubmit="return doValidate('form1')">
                            用户名:
                            <input type="text" name="username" size="20">
                            密  码:
                            <input type="password" name="password" size="20">
                            <input type="reset" value=" 取 消 ">
                        </form>
...
    </body>
...
```

注意：当使用 JSValidation 框架来实现客户端验证时，虽然工作量增加了，但是通过配置 XML 文件就可以实现相应的验证功能。

至此，就完成了 JSValidation 框架的使用。

5.3　服务器端验证

客户端校验证只能校验所设置校验的 form 表单元素，而对输入表单元素中的内容却不能做出很好的验证。例如判断输入的内容是否为 NULL，过滤一些关于特殊元素等。本节将通过服务器端程序对 form.htm 输入的内容进行验证。

5.3.1　校验输入字符工具类

首先创建一个名为 StringTool 的工具类，在该类中分别实现了 3 个方法：用来判断输入内容是否为空的方法 validateNull()、用来实现替换字符方法 chanageNull()和用来过滤特殊元素的方法 filterHtml()。

StringTool.java 是实现了校验输入字符的工具类，具体内容如代码 5.4 所示。

代码 5.4　校验字符串工具类：StringTool.java

```
public class StringTool {
    //判断输入的字符串参数是否为空
    public static boolean validateNull(String args) {
        if (args == null || args.length() == 0) {
            return true;
        } else {
            return false;
        }
    }
//判断输入的字符串参数是否为空或者是“null”字符，如果是，就返回 target 参数，如果不是，就返回源参数
    public static String chanageNull(String source, String target) {
        if (source == null || source.length() == 0 || source.
        equalsIgnoreCase("null")) {
            return target;
        } else {
            return source;
        }
    }
    //过滤<，>，\n 字符的方法
    public static String filterHtml(String input) {
        if (input == null) {
            return null;
        }
        if (input.length() == 0) {
            return input;
        }
        input = input.replaceAll("&", "&amp;");
        input = input.replaceAll("<", "&lt;");
        input = input.replaceAll(">", "&gt;");
        input = input.replaceAll(" ", " ");
        input = input.replaceAll("'", "&#39;");
        input = input.replaceAll("\"", "&quot;");
        return input.replaceAll("\n", "<br>");
    }
}
```

【代码解析】

- validateNull()方法用来验证输入的字符是否为 NULL 或为空。
- chanageNull()方法用来处理一些特殊情况：当输入的字符为空或 NULL（不区分大小写）时，就用第二个参数代替空或值为 NULL 的字符。
- filterHtml()方法用来过滤<、>、\n 等元素。

5.3.2　处理输入字符类

当 form.htm 页面发出请求后，服务器端的 CheckMessageServlet 首先会获取请求中传送过来的参数，然后通过调用 StringTool 类中的方法对参数进行处理，最后再显示出这些参数的值。

CheckMessageServlet.java 类实现对参数值的处理，具体内容如代码 5.5 所示。

代码 5.5　处理输入字符类：CheckMessageServlet.java

```
...
public class CheckMessageServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");          //设置编码格式
        String name = request.getParameter("name");     //获取 name 参数的值
        String title = request.getParameter("title");  //获取 title 参数的值
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>显示所填内容</title></head>");
        out.println("<body>");
        //通过 StringTool.validateNull()方法检验 name
        if (StringTool.validateNull(name)) {
            out.println("对不起，姓名不能为空，请您重新输入！<br>");
            out.println("<a href=\"" + request.getContextPath()
                    + "/form.htm\">添加新的留言</a><br>");
        //通过 StringTool.validateNull()方法检验 title
        } else if (StringTool.validateNull(title)) {
            out.println("对不起，主题不能为空，请您重新输入！<br>");
            out.println("<a href=\"" + request.getContextPath()
                    + "/form.htm\">添加新的留言</a><br>");
        } else {                                        //输出表单内容
            out.println("<table width=\"600\" border=\"1\" bordercolor=
            \"000000\" style=\"table-layout:fixed;word-break:
            break-all\">");
...
            //过滤掉 name 参数中的特殊元素
            out.println("姓名: "+ StringTool.filterHtml(name));
            //当电话为空时，用没填代替
            out.println("电话:+StringTool.chanageNull(request.
            getParameter("phone"), "没填"));
            //当 E-mail 为空时，用没填代替
            out.println("email:+ StringTool.chanageNull(request.
            getParameter("email"), "没填")");
            //过滤掉 title 参数中的特殊元素
            out.println("主题:+ StringTool.filterHtml("title")");
            //当内容为空时，用没填代替
            out.println("内容:+ StringTool.chanageNull(request.
            getParameter("content"), "没填")");
            out.println("</table><br>");
            out.println("</body>");
            out.println("</html>");
            out.flush();
            out.close();
        }
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 CheckMessageServlet 类路径-->
<servlet>
```

```
    <servlet-name>CheckMessageServlet</servlet-name>
    <servlet-class>com.cjg.servlet.CheckMessageServlet</servlet-class>
</servlet>
<!--配置 CheckMessageServlet 类映射-->
<servlet-mapping>
    <servlet-name>CheckMessageServlet</servlet-name>
    <url-pattern>/servlet/CheckMessage</url-pattern>
</servlet-mapping>
```

【代码解析】

- 在上述代码中，首先获取 name 和 title 参数中的值，然后通过 StringTool.validateNull()方法判断两者是否为空。之所以只检验这两个值，是因为客户端只限制两者不为空。
- 当 name 和 title 参数中的值不为空时，则可以对各个参数的值进行显示。对于 name 和 title 参数中的值，要通过 StringTool.filterHtml()方法过滤掉特殊元素；当显示参数 phone、E-mail 和 content 的值时，则调用 StringTool.chanageNull()方法，如果这些值为空，则需要显示为“没填”。

注意：对于大型网站，除了需要编写客户端验证外，还必须进行服务器端验证。

5.4　实现图形验证码

网站程序的安全是系统开发人员必须考虑的重要因素之一，如果考虑不全面就有可能会给系统的使用者和管理者带来严重问题。例如经常上网的人可以利用攻击软件（注册机等），通过扫描网页中的表单，而频繁发送相同信息造成不良的影响；或者不断地尝试盗取用户名和密码等。为了解决这些问题，可以使用验证码技术。

5.4.1　为什么要使用验证码技术

绝大多数 Web 站点一般都会碰到用户的恶意攻击，其中一种很常见的攻击手段就是身份欺骗。所谓身份欺骗可以通过在客户端利用脚本写入一些代码，然后利用该代码在网站、论坛等系统上反复登录；或者创建一个 HTML 窗体，该窗体首先包含了与所需注册或发帖窗体相同的字段，然后利用 http-post 传输数据到服务器，该服务器就会执行并创建相应账户，提交垃圾数据等操作。如果服务器本身不能通过有效验证拒绝这些非法操作，它们就会严重耗费服务器系统资源、降低网站性能甚至使服务器崩溃。

到目前为止，网站上比较流行的判断访问 Web 程序是合法用户还是恶意操作的技术，是采用了一种叫做“验证码”的技术。所谓验证码，就是首先将一串随机产生的数字或符号生成一幅图片，然后在图片中加入一些干扰像素，最后由用户肉眼识别其中的验证码信息，输入表单并提交网站验证，验证成功后才能使用相应功能。

于是大多数 Web 网站，经常会为客户端提供一个包含随机产生字符串的图片。浏览者只有读取这些字符串，然后在登录窗体或者发帖窗体等相应的地方填写这些字符串，才能使用相应功能。为什么必须在随机产生的字符串图片中加入一些干扰元素？因为只有浏览

者才可以很容易读出图片中的字符串，而如果是一段客户端攻击代码，通过一般手段是很难识别带有干扰元素的验证码。

随着时间的推移，验证码经历了 3 个过程。早期时使用数字即数字形式显示在网页上，恶意者可以通过复制粘贴来输入（或者直接使用软件来获取）；接着就出现规则数字图片，也就是直接用网页产生数字图片，这个就不能实现复制粘贴，但是恶意者使用 OCR 功能（就是图片文字识别技术）用软件来自动判断图片上的数字；现在一般使用的验证码，即对图片上显示的数字或文字进行了特殊处理，加了其他杂色，来干扰有 OCR 功能软件对这些验证码的读取。

虽然验证码技术很实用，但是在具体编写验证码时需要考虑许多方面。例如时间性：一般的验证码都需要设置时间的有效性，浏览者必须在失效前将注册、发表等操作完成；可恢复性：在有验证码的窗口有时需要添加许多文字，如果提交失败，那么这些文字很难保证能够返回。所以在提交带有验证码的窗口时，需要自动将浏览者填写的文字复制到剪贴板（系统自带的复制缓存）上。

5.4.2　图形验证码的具体实现

为了限制浏览者利用工具软件来暴力猜测密码，本节将通过在服务器端编程的方式来实现验证码。首先通过服务器端程序产生一个验证码，然后在登录页面显示出产生的验证码，最后由浏览者把验证码手工填写进相应的地方。为了不给浏览者增加太多输入麻烦，验证码不宜太长，通常为 4 个随机字符。具体步骤如下。

（1）首先编写一个登录页面，在该页面中显示出验证码图片，代码 5.6 用来显示验证码。

代码 5.6　显示验证码：code.html

```
...
<h3>带有验证码的登录页面</h3>
...
<form action="servlet/LogonFormServlet" method="post">
    用户名：<input type="text" name="name"><br>     <!--用户输入框-->
    密  码：<input type="password" name="pass"><br>  <!--密码框-->
    验证码：<input type="text" name="check code">
    <img src="servlet/CheckCodeServlet"><br>       <!--显示出验证码图片-->
    <input type="submit" value="登录">
</form>
...
```

（2）接着创建一个名为 CheckCodeServlet 的服务器端程序，代码 5.7 用来产生带有随机验证码的图片。

代码 5.7　产生验证码：CheckCodeServlet.java

```
...
public class CheckCodeServlet extends HttpServlet
{
    private static int WIDTH = 60;                      //定义图片的宽度
```

```
private static int HEIGHT = 20;                  //定义图片的高度
public void doGet(HttpServletRequest request,HttpServletResponse
response)
        throws ServletException,IOException
{
    HttpSession session = request.getSession(); //获取 HttpSession 对象
    response.setContentType("image/jpeg");      //设置编码
    ServletOutputStream sos = response.getOutputStream();  //获取输出流
    //设置浏览器不要缓存此图片
    response.setHeader("Pragma","No-cache");
    response.setHeader("Cache-Control","no-cache");
    response.setDateHeader("Expires", 0);
    //创建内存图像并获得其图形上下文
    BufferedImage image =
        new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE INT RGB);
    Graphics g = image.getGraphics();
    char [] rands = generateCheckCode();  //产生随机的认证码
    //产生图像
    drawBackground(g);
    drawRands(g,rands);
    g.dispose();                          //结束图像的绘制过程，完成图像
    //将图像输出到客户端
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    ImageIO.write(image, "JPEG", bos);
    byte [] buf = bos.toByteArray();
    response.setContentLength(buf.length);
    sos.write(buf);
    bos.close();
    sos.close();
    session.setAttribute("check code",new String(rands));
                                          //将当前验证码存入到 Session 中
}
//产生随机验证码
private char [] generateCheckCode()
{
    String chars = "0123456789abcdefghijklmnopqrstuvwxyz";
                                          //定义验证码的字符表
    char [] rands = new char[4];
    for(int i=0; i<4; i++)
    {
        int rand = (int)(Math.random() * 36);
        rands[i] = chars.charAt(rand);
    }
    return rands;
}
//输出没有干扰元素的验证码图片
private void drawRands(Graphics g , char [] rands)
{
    g.setColor(Color.BLACK);
    g.setFont(new Font(null,Font.ITALIC|Font.BOLD,18));
    //在不同的高度上输出验证码的每个字符
    g.drawString("" + rands[0],1,17);
    g.drawString("" + rands[1],16,15);
    g.drawString("" + rands[2],31,18);
    g.drawString("" + rands[3],46,16);
    System.out.println(rands);
}
//输出有干扰元素的验证码图片
```

```
    private void drawBackground(Graphics g)
    {
        //画背景
        g.setColor(new Color(0xDCDCDC));
        g.fillRect(0, 0, WIDTH, HEIGHT);
        //随机产生 120 个干扰点
        for(int i=0; i<120; i++)
        {
            int x = (int)(Math.random() * WIDTH);
            int y = (int)(Math.random() * HEIGHT);
            int red = (int)(Math.random() * 255);
            int green = (int)(Math.random() * 255);
            int blue = (int)(Math.random() * 255);
            g.setColor(new Color(red,green,blue));
            g.drawOval(x,y,1,0);
        }
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 CheckCodeServlet 类路径-->
<servlet>
    <servlet-name>CheckCodeServlet</servlet-name>
    <servlet-class>com.cjg.servlet.CheckCodeServlet</servlet-class>
</servlet>
<!--配置 CheckCodeServlet 类映射-->
<servlet-mapping>
    <servlet-name>CheckCodeServlet</servlet-name>
    <url-pattern>/servlet/CheckCodeServlet</url-pattern>
</servlet-mapping>
```

注意：上述类在具体编写时，除了 doGet()需要编写外，generateCheckCode()和 drawBackground()方法可以参考网上实现相应功能的方法。

（3）最后创建一个名为 LogonFormServlet 的服务器端程序，代码 5.8 用来处理登录表单的请求。

代码 5.8　处理请求：LogonFormServlet.java

```
...
public class LogonFormServlet extends HttpServlet
{
    public void service(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException
    {
        response.setContentType("text/html;charset=GB2312");//设置编码方式
        PrintWriter out = response.getWriter();             //获取输出流
        HttpSession session = request.getSession(false);
                                                    // 获取 Session 对象
        if(session == null)                         //判断 Session 对象
        {
            out.println("验证码处理问题!");
            return;
        }
        //获取 session 中存储 heck_code 对象的值
        String savedCode = (String)session.getAttribute("check_code");
```

```
        if(savedCode == null)                    //判断对象 savedCode
        {
            out.println("验证码处理问题!");
            return;
        }
        //获取请求信息中 check code 的值
        String checkCode = request.getParameter("check code");
        if(!savedCode.equals(checkCode))    //当请求与 Session 存储对象不相同
        {
            out.println("验证码无效!");
            return;
        }
        session.removeAttribute("check code");
                                        //移除 Session 对象中 check_code 的值
        out.println("验证码通过，服务器正在校验用户名和密码!");
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 LogonFormServlet 类路径-->
<servlet>
    <servlet-name>LogonFormServlet</servlet-name>
    <servlet-class>com.cjg.servlet.LogonFormServlet</servlet-class>
</servlet>
<!--配置 LogonFormServlet 类映射-->
<servlet-mapping>
    <servlet-name>LogonFormServlet</servlet-name>
    <url-pattern>/servlet/LogonFormServlet</url-pattern>
</servlet-mapping>
```

【代码解析】

在上述代码中，首先判断请求和 Session 是否为空，当为空时则会输出“验证码处理问题”。如果都不为空，则获取请求和 Session 对象中 check_code 的值，然后比较这两个对象是否相同。当相同时，则移除 Session 中 check_cod，然后输出“验证码通过，服务器正在校验用户名和密码!”信息，否则就会输出“验证码无效!”的信息。

5.5　避免重复提交功能

之所以会出现重复提交现象，是由于当服务器端负荷过重或网络拥挤而出现响应缓慢情况时，浏览者在提交 Form 表单没有立即看到服务器端的响应后，于是就会怀疑提交没有成功而再次单击“提交”按钮，最终就会发生同一份表单重复提交。本节中将详细介绍在客户端如何避免表单的重复提交，以及在服务器端如何避免表单的重复提交。

5.5.1　客户端避免重复提交

对于有些行业特别是金融行业等，重复提交同一笔交易是绝对不应该出现的，所以网站开发人员必须想办法禁止表单的重复提交。在客户端可以通过 JavaScript 脚本语言来控制表单提交的次数。

下面将通过一个具体的实例来详细介绍如何利用 JavaScript 脚本语言，避免表单的重复提交，具体步骤如下。

（1）首先编辑 index.jsp 页面，在该页面中实现登录功模块，具体内容如代码 5.9 所示。

代码 5.9　登录页面：index.jsp

```
...
   <head>
      <title>登录页面</title>
      <!--避免重复提交功能-->
      <script language="javascript">
         <!--
            var checkSubmitFlg=true;          //定义了一个变量
            function checkSubmit()            //避免重复提交函数
            {
               if(true==checkSubmitFlg)       //判断变量的值
               {
                  document.theForm.submit(); //提交表单
                  checkSubmitFlg=false;      //设置变量的值为 false
               }
               else
               {
                  alert("你已经提交了表单，请不要重复提交！");
               }
            }
         -->
      </script>
   </head>
   <body>
      <form method="post" action="handler" name="theForm">
         <table>
            <tr>
               <td>用户名：</td>
               <td><input type="text" name="username"></td>
            </tr><tr>
               <td>密码：</td>
            <td>
               <input type="password" name="password">
            </td></tr><tr>
            <td><input type="reset" value="重填"></td>
            <!--调用函数 checkSubmit()-->
            <td><input type="button" name="btnSubmit" value="提交"
           onClick="checkSubmit();"/></td></tr>
         </table>
      </form>
   </body>
...
```

【代码解析】

函数 checkSubmit()除了可以如上述代码编写外，还可以编写成如下代码，即在提交表单时同时将“提交”按钮变灰，禁止其再次使用。

```
function checkSubmit()
{
   if(true==checkSubmitFlg)
   {
```

```
        document.theForm.btnSubmit.disabled=true;   //“提交”按钮变灰
        document.theForm.submit();
        checkSubmitFlg=false;
    }
}
```

（2）为了能够更好地演示对重复提交的控制，下面将创建一个 HandlerServlet 类，代码 5.10 模拟服务器端响应缓慢的功能。

代码 5.10　控制重复提交：HandlerServlet.java

```
...
public class HandlerServlet extends HttpServlet {
    int count = 0;                          //定义了一个变量用来显示访问次数
    public void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        resp.setContentType("text/html;charset=GBK");
        PrintWriter out = resp.getWriter();         //获取输出流
        try {
            Thread.sleep(4000);                     //让线程休眠 4 秒
        } catch (InterruptedException e) {
            System.out.println(e);
        }
        System.out.println("submit : " + count);   //打印出访问次数 count
        if (count % 2 == 1)                         //判断 count 变量
            count = 0;
        else
            count++;
        out.println("success");                     //输出是否为重复提交
        out.close();
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 HandlerServlet 类路径-->
<servlet>
    <servlet-name>HandlerServlet</servlet-name>
    <servlet-class>com.cjg.servlet.HandlerServlet</servlet-class>
</servlet>
<!--配置 HandlerServlet 类映射-->
<servlet-mapping>
    <servlet-name>HandlerServlet</servlet-name>
    <url-pattern>/handler</url-pattern>
</servlet-mapping>
```

【代码解析】

在上述代码中通过调用 Thread.sleep()方法，让当前线程睡眠 4 秒，即通过刻意延长 4 秒来模拟服务器端响应缓慢的情况。同时通过变量 count，在服务器端记录客户端提交的次数。

最后发布该项目后，通过浏览 http://localhost:8080/redundantsub/index.jsp 地址就会显示出登录界面。填写相应的信息后，如果连续单击两次“提交”按钮，就会弹出如图 5.21 所示的页面。这时单击“确定”按钮，就会弹出如图 5.22 所示的显示是否提交成功页面。虽然在客户端提交了两次，但是由于 JavaScript 语言的控制，从开发环境的输出窗口中看到

却是提交了一次，如图 5.23 所示。



图 5.21　两次单击



图 5.22　输出窗口

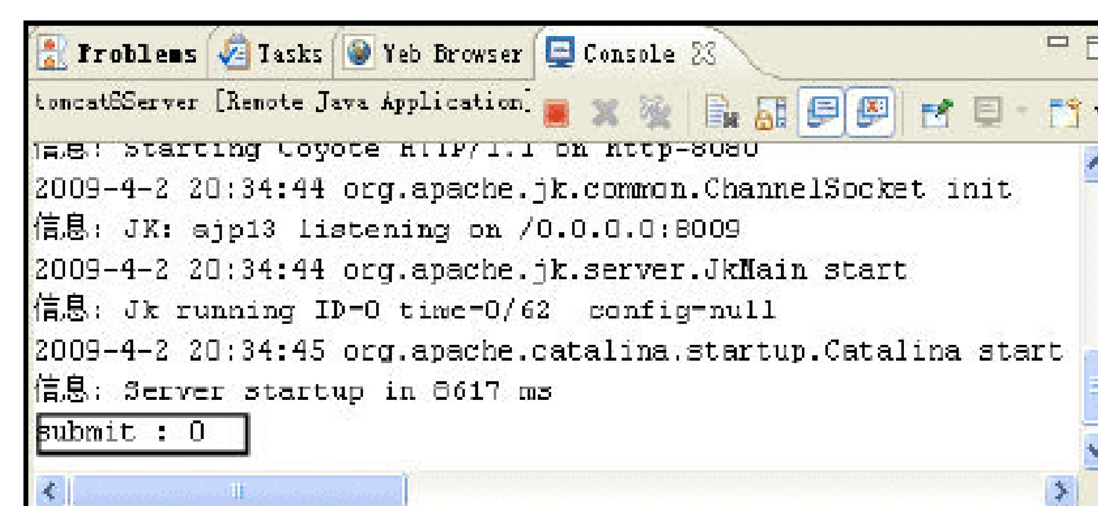


图 5.23　提交次数

5.5.2　服务器端避免重复提交

在客户端通过 JavaScript 脚本控制表单的重复提交时，存在不足之处。即当显示提交成功后，浏览者如果单击“刷新”按钮，将导致表单再次提交；或者浏览者单击“后退”按钮后，也会导致表单的重复提交。

为了解决上述的不足，可以在服务器端编写相应的程序避免表单重复提交，其基本原理如下：

index.jsp 页面中部分内容是由服务器端程序 TokenProcessor 动态产生，具体过程就是 TokenProcessor 会为每次产生的包含 Form 表单页面，产生一个唯一的随机标识号，该标识号不仅要设置成为隐藏域中的值，而且还必须保存到 Session 域。

当浏览者提交 Form 表单时，HandlerServlet 程序会比较隐藏域中的值与 Session 域中的标识号，如果相同则处理表单数据，处理后就清除当前用户 Session 域中存储的标识号，如果不相同，则会忽略表单请求。当重复提交相同 From 表单页面时，当前 Session 域中会不存在相应的表单标识号。

（1）首先创建一个名为 TokenProcessor 的服务器端程序，代码 5.11 用来产生令牌。

代码 5.11　令牌类：TokenProcessor.java

```
...
public class TokenProcessor
{
    static final String TOKEN KEY="org.sunxin.token";
    private static TokenProcessor instance = new TokenProcessor();
    //获得该类的实例
    public static TokenProcessor getInstance()
    {
        return instance;
```

```
}
private long previous;              //最近一次生成令牌值的时间戳
//判断请求参数中的令牌值是否有效
public synchronized boolean isTokenValid(HttpServletRequest request)
{
    HttpSession session = request.getSession(false);
                                    //得到请求的当前 Session 对象
    if (session == null)
    {
        return false;
    }
    //从 Session 中取出保存的令牌值
    String saved = (String) session.getAttribute(TOKEN_KEY);
    if (saved == null) {
        return false;
    }
    resetToken(request);            //清除 Session 中的令牌值
    String token = request.getParameter(TOKEN_KEY);
                                    //得到请求参数中的令牌值
    if (token == null) {
        return false;
    }
    return saved.equals(token);
}
//清除 Session 中的令牌值
public synchronized void resetToken(HttpServletRequest request)
{
    HttpSession session = request.getSession(false);
    if (session == null) {
        return;
    }
    session.removeAttribute(TOKEN_KEY);
}
//产生一个新的令牌值，保存到 Session 中
public synchronized void saveToken(HttpServletRequest request)
{
    HttpSession session = request.getSession();
    String token = generateToken(request);
    if (token != null) {
        session.setAttribute(TOKEN_KEY, token);
    }
}
//根据用户会话 ID 和当前的系统时间生成一个唯一的令牌
public synchronized String generateToken(HttpServletRequest request)
{
    HttpSession session = request.getSession();
    try
    {
        byte id[] = session.getId().getBytes();
        long current = System.currentTimeMillis();
        if (current == previous)
        {
            current++;
        }
        previous = current;
        byte now[] = new Long(current).toString().getBytes();
        MessageDigest md = MessageDigest.getInstance("MD5");
        md.update(id);
        md.update(now);
        return toHex(md.digest());
```

```
        }
        catch (NoSuchAlgorithmException e)
        {
            return null;
        }
    }
    //将一个字节数组转换为一个十六进制数字的字符串
    private String toHex(byte buffer[])
    {
        StringBuffer sb = new StringBuffer(buffer.length * 2);
        for (int i = 0; i < buffer.length; i++)
        {
            sb.append(Character.forDigit((buffer[i] & 0xf0) >> 4, 16));
            sb.append(Character.forDigit(buffer[i] & 0x0f, 16));
        }
        return sb.toString();
    }
    //从 Session 中得到令牌值，如果 Session 中没有保存令牌值，则生成一个新的令牌值
    public synchronized String getToken(HttpServletRequest request)
    {
        HttpSession session = request.getSession(false);
        if(null==session)
            return null;
        String token=(String)session.getAttribute(TOKEN_KEY);
        if(null==token)
        {
           token = generateToken(request);
            if (token != null)
            {
                session.setAttribute(TOKEN_KEY, token);
                return token;
            }
            else
                return null;
        }
        else
            return token;
    }
}
```

注意：编写上述代码时，可以查看 Struts 框架中的一个同步令牌类（org.apache.struts.util.TokenProcessor），该类封装了对同步令牌进行处理的方法。

（2）接着创建一个 index.jsp 页面，在该页面中除了一些必要的内容外，还添加一个隐蔽输入域，代码 5.12 为 index.jsp 页面的源代码。

代码 5.12　带有隐蔽域的页面：index.jsp

```
...
<%@ page import="com.cjg.servlet.TokenProcessor" %>
...
    <body>
        <%
            //获取令牌类实例
            TokenProcessor processor=TokenProcessor.getInstance();
            //获取令牌值
            String token=processor.getToken(request);
        %>
        <form method="post" action="handler" name="theForm">
```

```
        <table>
            <tr>
                <td>用户名：</td>
                <td><input type="text" name="username"></td>
            </tr>
            <tr>
                <td>密码：</td>
             <td>
                 <input type="password" name="password">
                 <!--设置隐藏域，其值为令牌值-->
                 <input type="hidden" name="org.sunxin.token" value=
                 "<%=token%>"/>
             </td>
         </tr>
         <tr>
             <td><input type="reset" value="重填"></td>
             <td><input type="button" name="btnSubmit" value="提交"
              onClick="checkSubmit();"/></td>
         </tr>
         </table>
     </form>
...
```

注意：在上述代码中，增加一个隐藏输入域，该域的值为服务器端产生的令牌值。隐藏输入域的名字必须与 TokenProcessor 类中定义的静态常量 TOKEN_KEY 的值相同。

（3）修改 HandlerServlet 类，在该类中比较 Session 与隐藏域中的标识号。代码 5.13 控制重复提交。

代码 5.13　控制重复提交：HandlerServlet.java

```
...
public class HandlerServlet extends HttpServlet
{
    int count=0;
    public void doPost(HttpServletRequest req, HttpServletResponse resp)
              throws ServletException,IOException
    {
...//省略代码参考光盘源代码中此文件下的内容
        TokenProcessor processor=TokenProcessor.getInstance();
        if(processor.isTokenValid(req))
        {
...
        else
        {
            processor.saveToken(req);
            out.println("你已经提交了表单，同一表单不能提交两次。");
        }
        out.close();
    }
}
```

【代码解析】

在对提交的数据进行处理前（即调用 Thread.sleep()前），首先判断请求参数中的令牌值是否有效，如果有效，则进行下一步的处理，此时 Session 中的令牌值已在 isTokenValid()

方法中被移除。当用户重复提交时，isTokenValid()方法将会返回 false，同时输出“你已经提交了表单，同一表单不能提交两次。”的提示信息。如果浏览者再一次访问提交页面时，则会在隐藏输入域和 Session 中将重新保存相同的新的令牌值，这样就会利用同步令牌机制有效的解决重复提交的问题。

最后发布该项目后，通过浏览 http://localhost:8080/redundantsubservice/index.jsp 地址就会显示出登录界面，填写相应的信息后（如图 5.24 所示），只要单击“提交”按钮就会显示 success 信息（如图 5.25 所示）。如果浏览者单击“刷新”按钮，就会出现如图 5.26 所示的页面；如图浏览者单击“后退”按钮退到登录界面，在该界面填写相应的信息后单击“提交”按钮也会出现如图 5.26 所示的页面。



图 5.24　填写信息

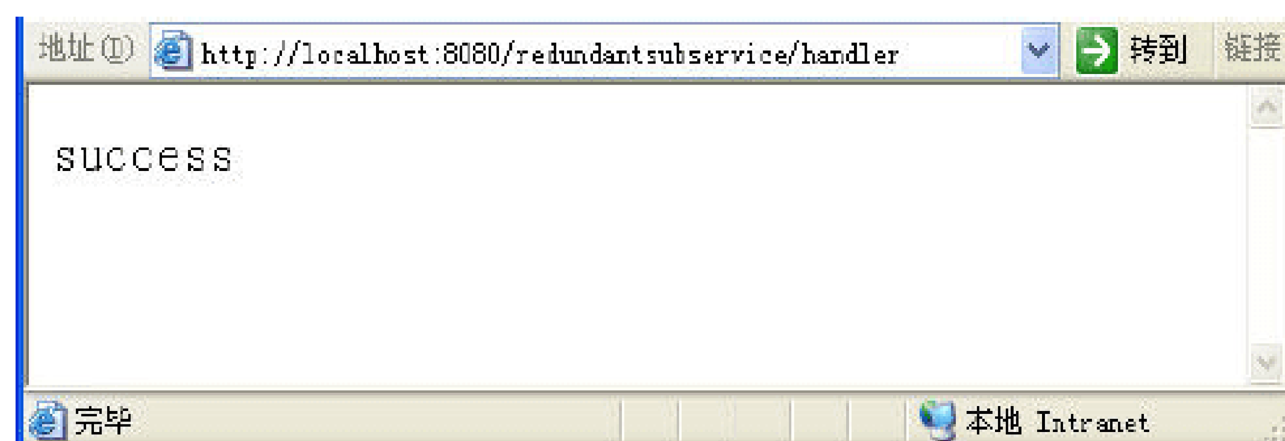


图 5.25　成功页面

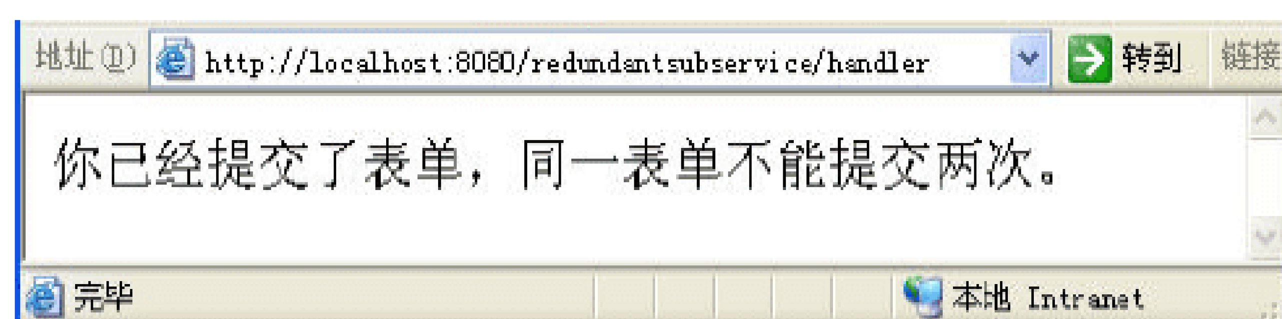


图 5.26　重复提交

5.6　缩略加水印图像

在浏览网页的时候，经常会使用缩略加水印的图像。之所以会这样，是因为网页会受到版面限制，而图像实际尺寸可能要比显示的尺寸大得多。同时为了防止别人盗用精心设计的图片，还需要对图像加上水印。

缩略加水印图像应用是一个非常实用的功能，在大型网站的页面上经常会出现其影子。本章为了便于讲解，只针对一张图片实现缩略加水印功能。

5.6.1　缩略加水印图像应用框架分析

对于一个大型网站系统来说，实现一个可用的缩略加水印图像应用功能要考虑的情况十分复杂，例如图像的类型、图像处理后的效果等。为了便于讲解，该节只对一张图片实现可用的缩略加水印图像，读者可以根据自己的需求自行进行完善。

在 Picture 项目中，image.html 页面会向服务器发出一个图片请求，在服务器端接受到该请求后就首先会找到所请求的图片，然后经过缩略加水印处理后，最后才会返回给浏览

器显示在 image.html 页面，具体过程如图 5.27 所示，该项目的目录结构如图 5.28 所示。

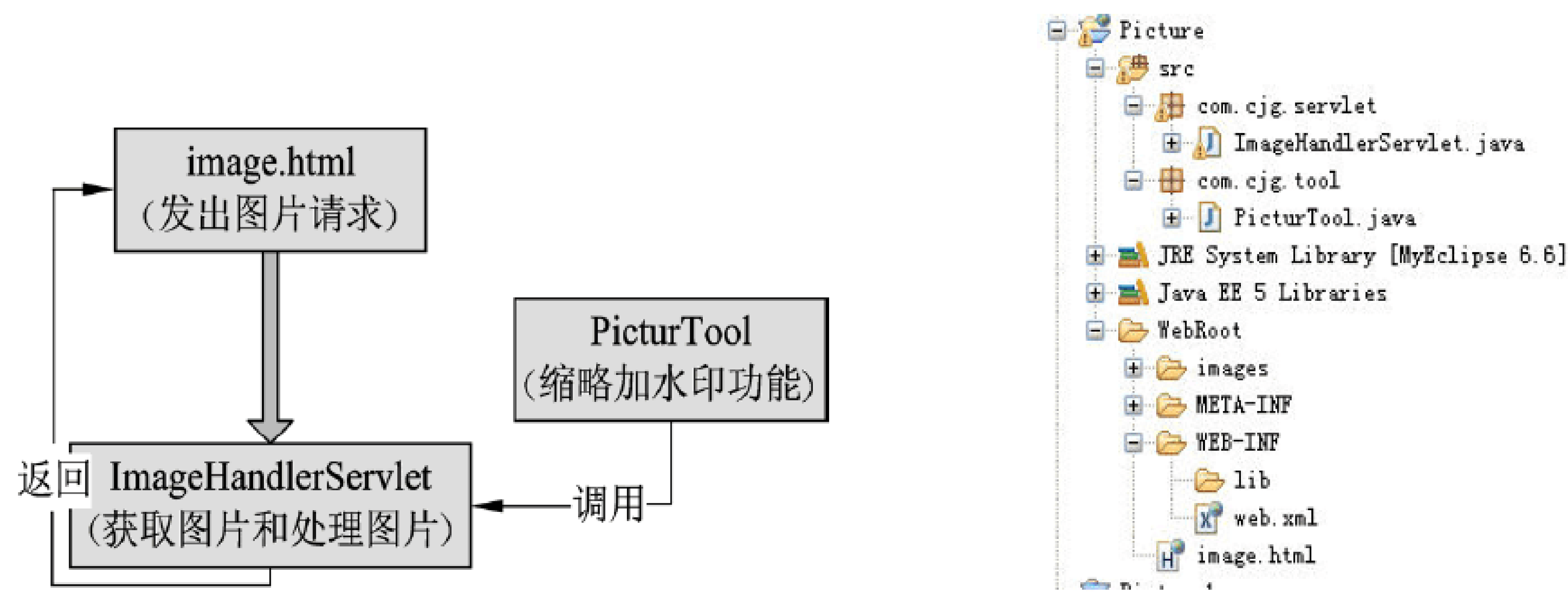


图 5.27　Picture 项目流程图　　　　　　　图 5.28　目录结构

当浏览者浏览 image.html 页面时，就会出现如图 5.29 所示的页面。在该页面中图片不仅被缩小而且在该图片上还加上了 cjg 的水印。

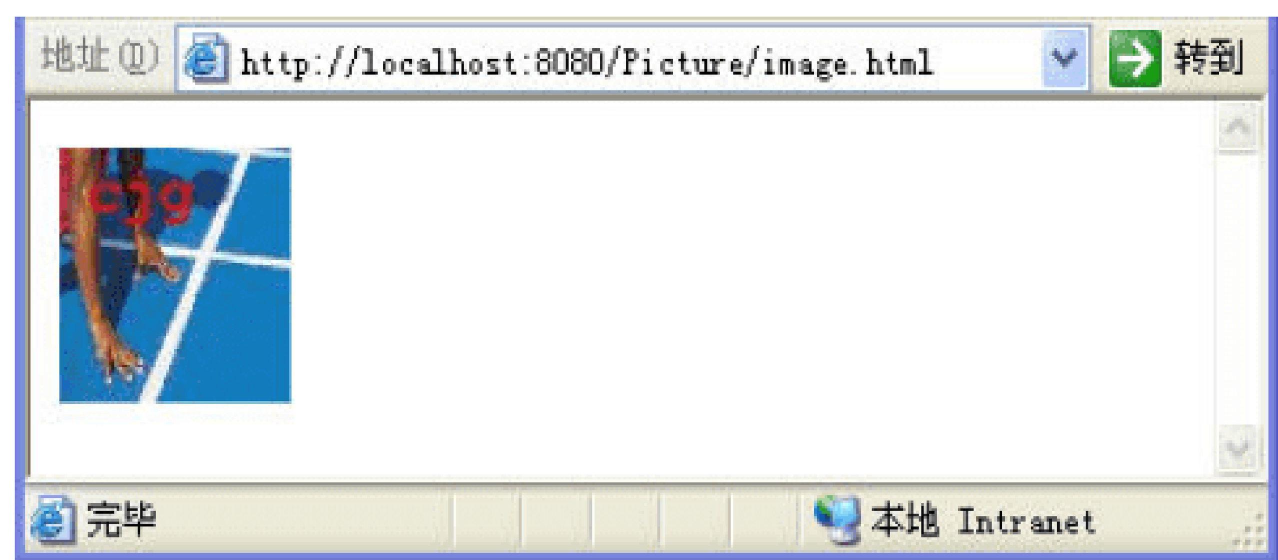


图 5.29　浏览图片

5.6.2　实现缩略加水印工具类

image.html 页面发出图片的请求后，服务器就会调用工具类来处理相应的图片，该工具类不仅实现图片的缩略功能而且还实现了加水印功能，具体内容如代码 5.14 所示。

代码 5.14　缩略加水印：PicturTool.java

```
...
public class PicturTool {
    //实现缩略功能
    public static BufferedImage abbreviatedzoom(String srcFileName,
            int outputWidth, int outputHeight) {
        //使用源图像文件名创建 ImageIcon 对象
        ImageIcon imgIcon = new ImageIcon(srcFileName);
        Image img = imgIcon.getImage();          //得到 Image 对象
        return abbreviatedzoom(img, outputWidth, outputHeight);
    }
    //实现缩略功能
    public static BufferedImage abbreviatedzoom(Image srcImage,
            int outputWidth, int outputHeight) {
        //构造一个预定义图像类型的 BufferedImage 对象
        BufferedImage buffImg = new BufferedImage(outputWidth,
        outputHeight,
                BufferedImage.TYPE_INT_RGB);
        //创建 Graphics2D 对象，用于在 BufferedImage 对象上绘图
        Graphics2D g = buffImg.createGraphics();
```

```
        g.setColor(Color.WHITE);        //设置图形上下文的当前颜色为白色
        //用图形上下文的当前颜色填充指定的矩形区域
        g.fillRect(0, 0, outputWidth, outputHeight);
        //按照缩放的大小在 BufferedImage 对象上绘制原始图像
        g.drawImage(srcImage, 0, 0, outputWidth, outputHeight, null);
        g.dispose();                    //释放图形上下文使用的系统资源
        return buffImg;
    }
    //实现加水印方法
    public static BufferedImage watermarkzoom(BufferedImage buffImg) {
        Graphics g = buffImg.getGraphics();     //获取 Graphics 对象
        g.setColor(Color.RED);                  //设置颜色
        Font font = new Font("Courier New", Font.BOLD, 20); //创建新的字体
        g.setFont(font);                    //设置图形上下文的字体为指定的字体
        //在图片上绘制文字，文字的颜色为图形上下文的当前颜色，即红色
        g.drawString("cjg", 10, 20);
        g.dispose();                        //释放图形上下文使用的系统资源
        return buffImg;
    }
}
```

【代码解析】

- ❑ 在上述代码中首先实现缩略方法，即根据指定的大小对图像进行缩小和放大。abbreviatedzoom(srcFileName,outputWidth,outputHeight)方法接受图像文件的名字，abbreviatedzoom(srcImage,outputWidth, outputHeight)方法接受 java.awt.Image 对象。
- ❑ watermarkzoom(buffImg)方法用来实现加水印功能，由于在该项目中实现缩略加水印功能时，先实现缩略功能后实现加水印，所以可以获取缩略后的 BufferedImage 对象然后在该画布上实现加水印功能。

5.6.3　对图像实现缩略加水印

服务器接受到请求后，就会调用 ImageHandlerServlet.java 代码来处理请求。该代码通过请求参数获取相应的图像，然后通过调用 5.6.2 节的工具类对该图像实现缩略加水印，具体内容如代码 5.15 所示。

代码 5.15　实现缩略加水印：ImageHandlerServlet.java

```
...
public class ImageHandlerServlet extends HttpServlet
{
   protected void service(HttpServletRequest req, HttpServletResponse
   resp)
                 throws ServletException, java.io.IOException
   {
      String strId=req.getParameter("id");        //获取请求中参数 ID 的值
      if(null==strId || "".equals(strId))         //判断参数 ID
      {
         throw new ServletException("图像参数错误！");
      }
      int id=Integer.parseInt(strId);             //转换 ID 值的类型
      String srcImgFileName=null;               //创建用来表示图像名称的变量
```

```
        //根据参数 id 的值设置 srcImgFileName 的值
        switch(id)
        {
        case 1:
            srcImgFileName=getServletContext().getRealPath("/")+"images/
            1.jpg";
            break;
        case 2:
            break;
        default:
            throw new ServletException("图像参数错误！");
        }
        resp.setContentType("image/jpeg");                  //设置返回类型
        ServletOutputStream sos=resp.getOutputStream();     //获取输出流
        //调用 PicturTool 类的静态方法 abbreviatedzoom(srcFileName,
        outputWidth, outputHeight)对原始图像进行缩放
        BufferedImage buffImg=PicturTool.abbreviatedzoom(srcImgFileName,
        80, 80);
        //调用 PicturTool 类的静态方法 watermarkzoom(buffImg)对缩放后图像进行加
        水印
        BufferedImage buffImg2=PicturTool.watermarkzoom(buffImg);
        //创建 JPEG 图像编码器，用于编码内存中的图像数据到 JPEG 数据输出流
        JPEGImageEncoder jpgEncoder=JPEGCodec.createJPEGEncoder(sos);
        //编码 BufferedImage 对象到 JPEG 数据输出流
        jpgEncoder.encode(buffImg2);
        sos.close();                                        //关闭输出流
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 ImageHandlerServlet 类路径-->
<servlet>
    <servlet-name>ImageHandlerServlet</servlet-name>
    <servlet-class>com.cjg.servlet.ImageHandlerServlet</servlet-
    class>
</servlet>
<!--配置 ImageHandlerServlet 类映射-->
<servlet-mapping>
    <servlet-name>ImageHandlerServlet</servlet-name>
    <url-pattern>/image</url-pattern>
</servlet-mapping>
```

【代码解析】

- 在上述代码中，首先获取请求中的 ID 号，然后根据 ID 号获取请求图像的路径。在该项目中为了便于讲解，利用 switch/case 语句写死了请求图像的路径。在实际开发项目中，如果图像数据是保存在数据库中，可以根据请求的参数就可以取出相应的图像。如果存储在硬盘上，就需要为所有的图片做一个索引文件，然后根据请求参数查找索引文件就可以得到图像路径。
- 获取到原图像后，首先通过调用工具类的 watermarkzoom(buffImg)方法实现缩略功能，然后通过调用工具类的 watermarkzoom(buffImg)方法实现加水印功能。

为了能够发出请求和显示图像处理后的效果，在该项目中创建了一个名为 image.html 的页面，具体内容如代码 5.16 所示。

代码 5.16　显示图像：image.html

```
...
    <body>
        <img src="image?id=1">           <!--链接相关图片-->
    </body>
...
```

5.7　小　　结

本章主要介绍验证模块：客户端表单验证、服务器端验证和图形验证码，同时还介绍了如何避免重复提交和实现图像的缩略加水印。

在具体实现验证模块时，首先通过调用 JSValidation 框架实现客户端验证，然后通过 JSP+Servlet 技术实现服务器端验证，同时还通过 JSP+Servlet 技术实现图形验证码。

本章中除了实现验证模块，还详细介绍避免重复提交功能：客户端避免重复提交和服务器端避免重复提交。最后，还实现了图像的缩略加水印功能。

第 6 章　网络硬盘（JSP+Servlet）

对于大型网络系统来说，有时候需要实现网络硬盘功能，使得用户可以随时上传文件、管理文件和下载文件。

本章将详细介绍实现网络硬盘功能的各个方面：如何浏览磁盘、如何浏览文件和文件夹，以及如何创建、查询和删除文件和文件夹。由于文件上传功能在后面章节将详细介绍，所以本章将不涉及上传文件功能。

6.1　网络硬盘功能原理

利用网络硬盘功能可以轻松地查看服务器空间中用户上传文件的各种信息，同时也可以轻松地在服务器端创建、删除文件夹等操作。该功能模块与文件上传和文件下载功能相结合可以实现强大的功能。

6.1.1　网络硬盘框架分析

对于一个大型网上系统来说，实现一个可用的网络硬盘功能要考虑的情况十分复杂，例如如何让浏览者查看允许范围内的文件目录？如何限制浏览者创建文件的类型？如何限制浏览者删除特定文件的权限等。该项目目录如图 6.1 所示。

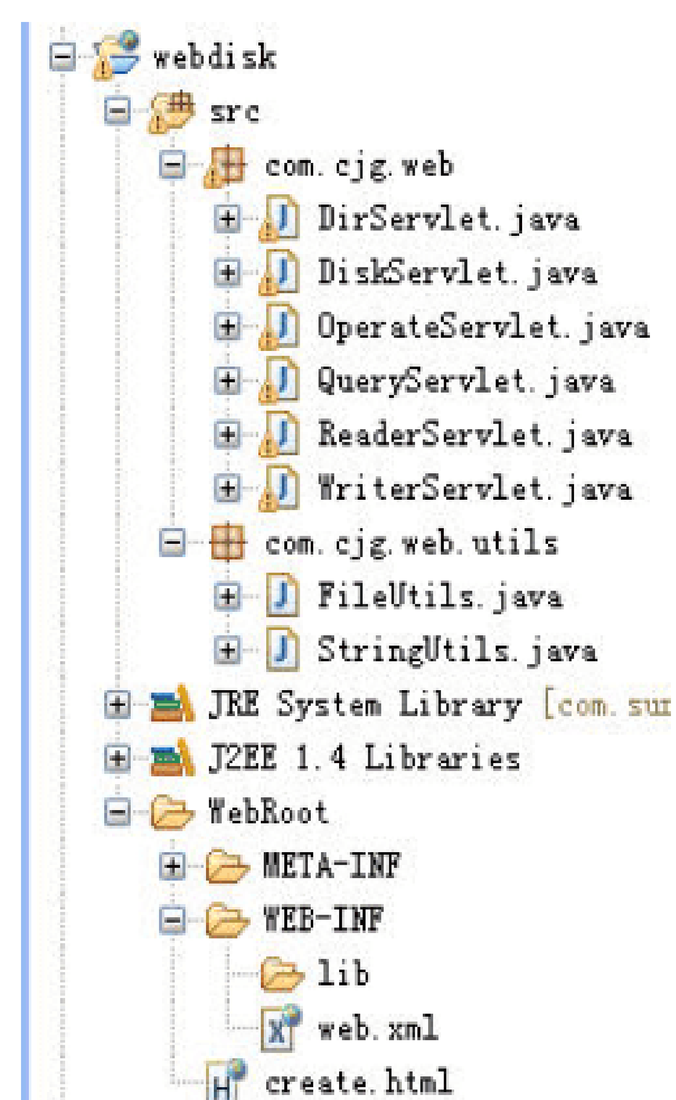


图 6.1　目录结构

6.1.2　网络硬盘功能描述

在本节中，以直观的方式来向读者介绍网络硬盘可以实现的功能。这些功能包括浏览磁盘和文件夹文件、创建文件和文件夹、显示文件内容和文件夹、查询特定文件和操作文件或文件夹。

1．浏览磁盘和文件夹文件信息

首先通过访问地址 http://localhost:8088/webdisk/DirServlet.shtml 显示磁盘列表，如图 6.2 所示。单击“服务器磁盘（D:\）”链接，则会显示该磁盘里的文件和文件夹列表，如图 6.3 所示。

2．创建文件和文件夹

在图 6.3 中单击“[创建文件/文件夹]”链接，就会出现如图 6.4 所示的创建文件或文件

夹对话框。如果想创建文件，首先选择“文件”单选按钮，然后输入相应的文件名 test.txt 和文件内容“test！！！”（如图 6.5 所示），最后单击“创建”按钮就可以转到显示 D 盘文件信息的页面（如图 6.6 所示）同时在该页面就会显示出新创建的 test.txt 文件。

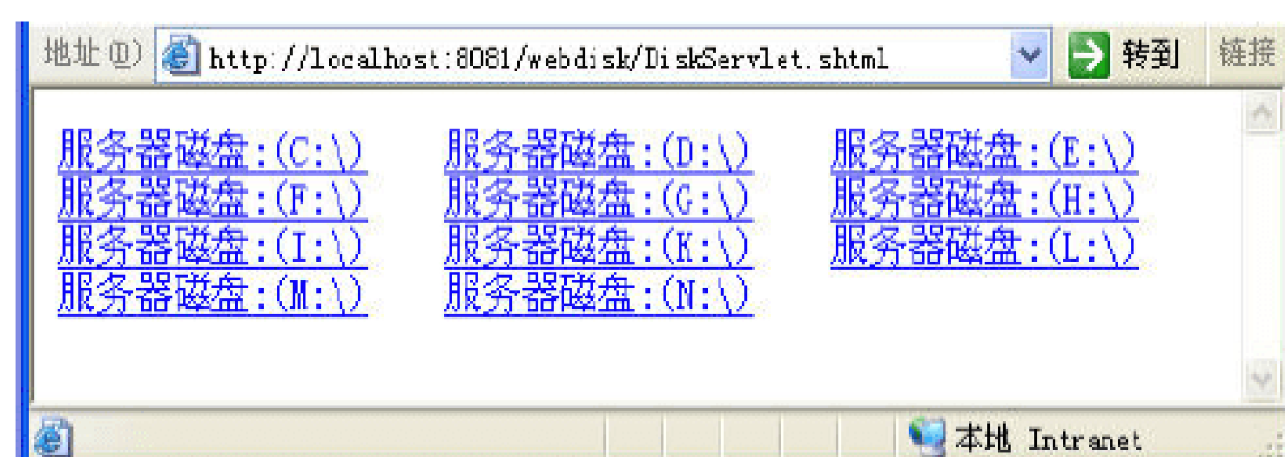


图 6.2　显示磁盘信息



图 6.3　显示 D 盘信息

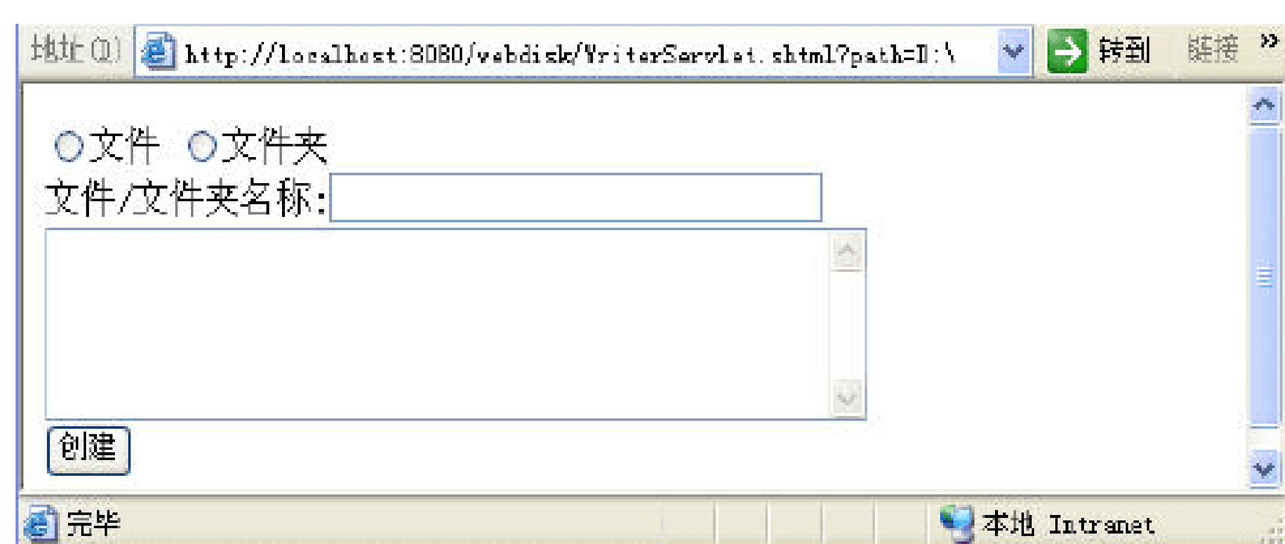


图 6.4　创建文件或文件夹对话框

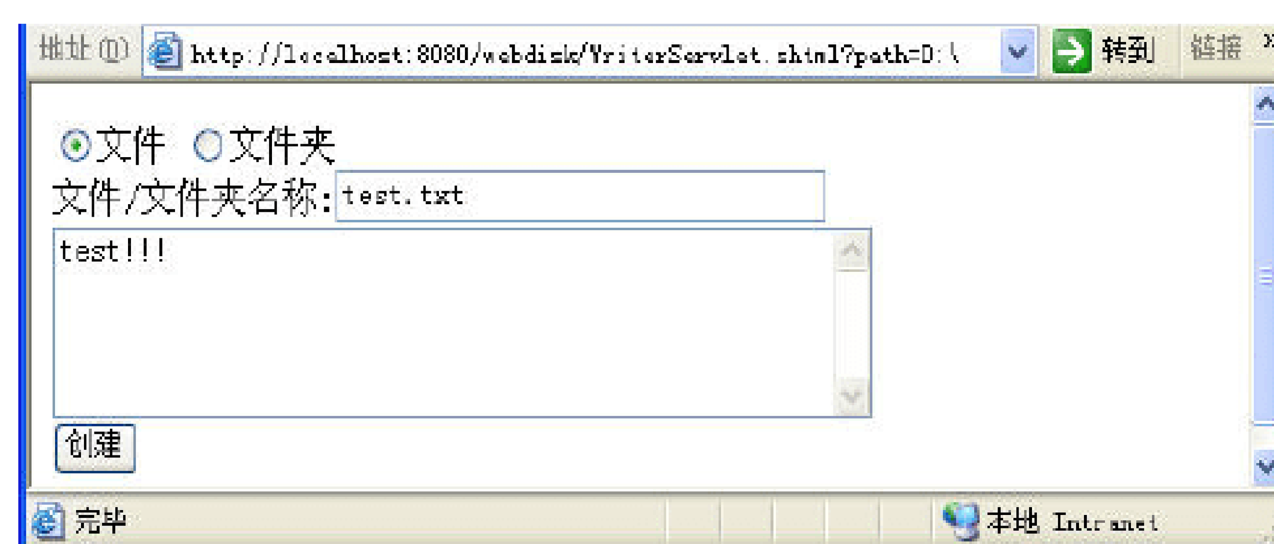


图 6.5　创建 test 文件



图 6.6　显示出 test.txt 文件

如果想创建文件夹，首先选择“文件夹”单选按钮，然后输入相应的文件夹名 test（如图 6.7 所示），最后单击“创建”按钮就可以转到显示 D 盘文件信息的页面（如图 6.8 所示），同时在该页面就会显示出新创建的 test 文件夹。

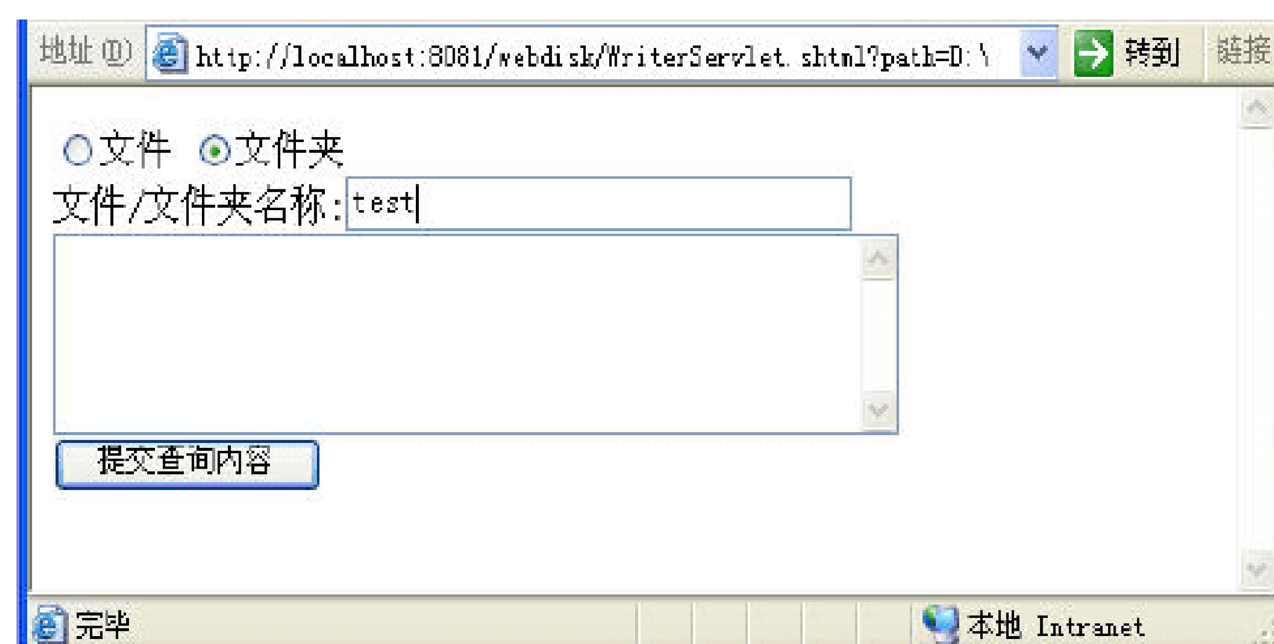


图 6.7　创建文件夹

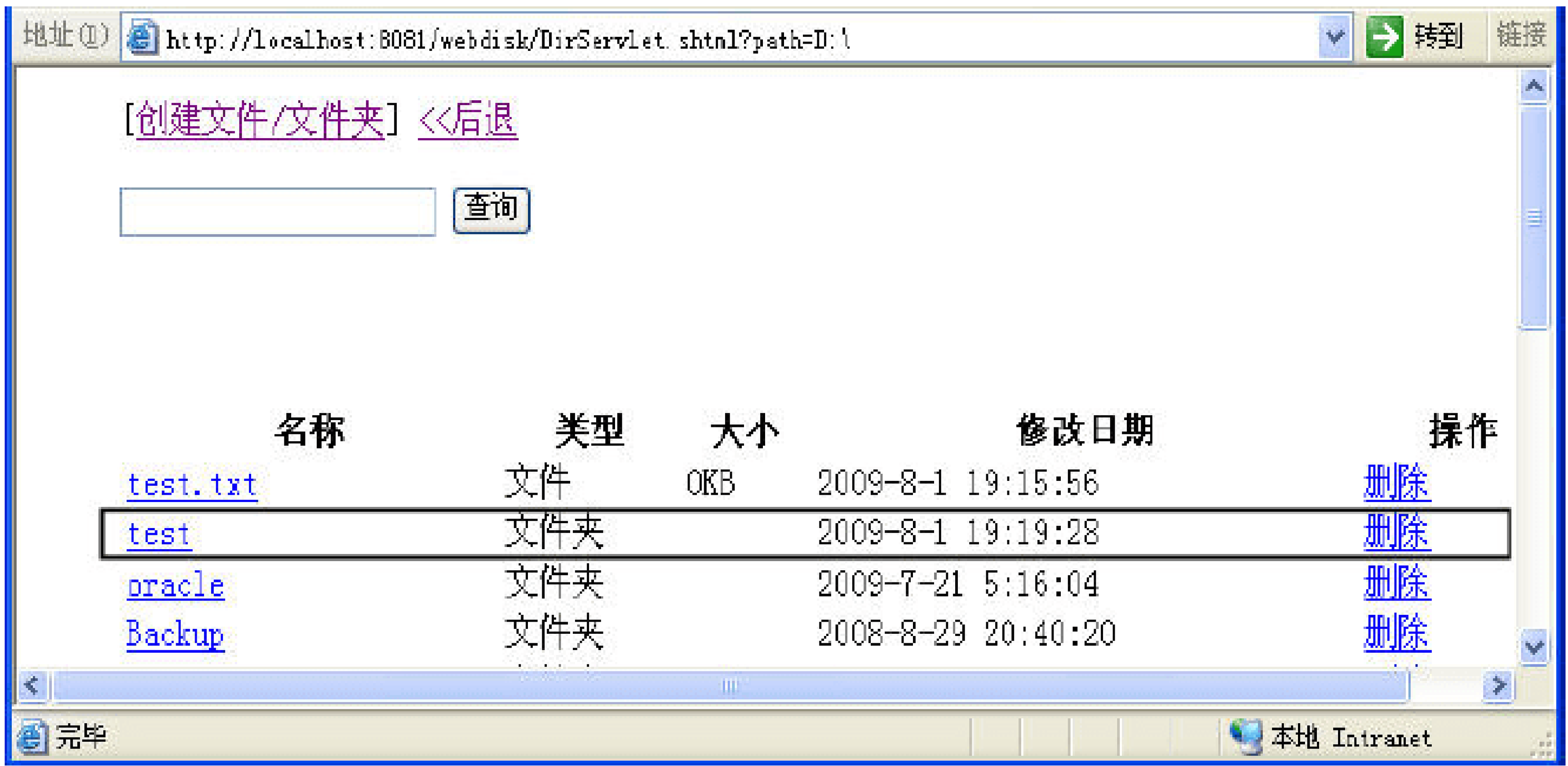


图 6.8　显示出 test 文件夹

3. 显示文件内容和文件夹

如果想查看文件的内容，可以在图 6.6 中单击 test.txt 链接，就会出现如图 6.9 所示的显示文件内容的页面。如果想查看文件夹中的文件列表，可以在图 6.8 中单击 test 链接，就会出现如图 6.10 所示的显示该文件夹中文件列表的页面。在该页面单击“后退”链接就会转到上一级目录，如图 6.11 所示。

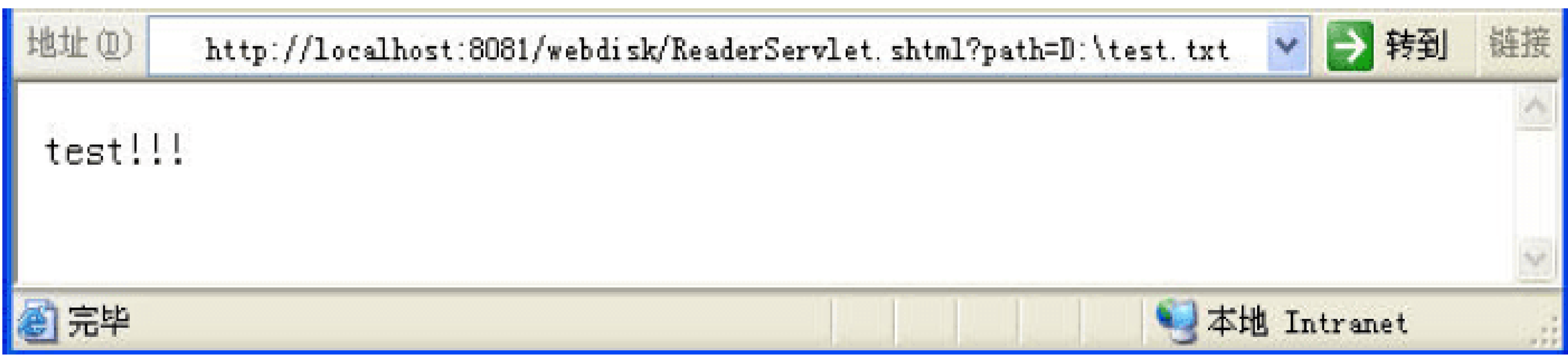


图 6.9　显示文件内容



图 6.10　显示文件夹列表



图 6.11　上一级目录

4．查询特定文件

当文件特别多时，如果想查找特定的文件或文件夹，可以使用查询功能。例如可以在文本框中输入文件名 test 字符串（如图 6.12 所示），然后单击“查询”按钮就可以出现关于该名称文件的信息，如图 6.13 所示。为了方便于后面的讲解，在 text 文件夹中再创建一个名为 testfile.txt 文件，具体信息如图 6.14 所示。



图 6.12　查询信息



图 6.13　查询结果

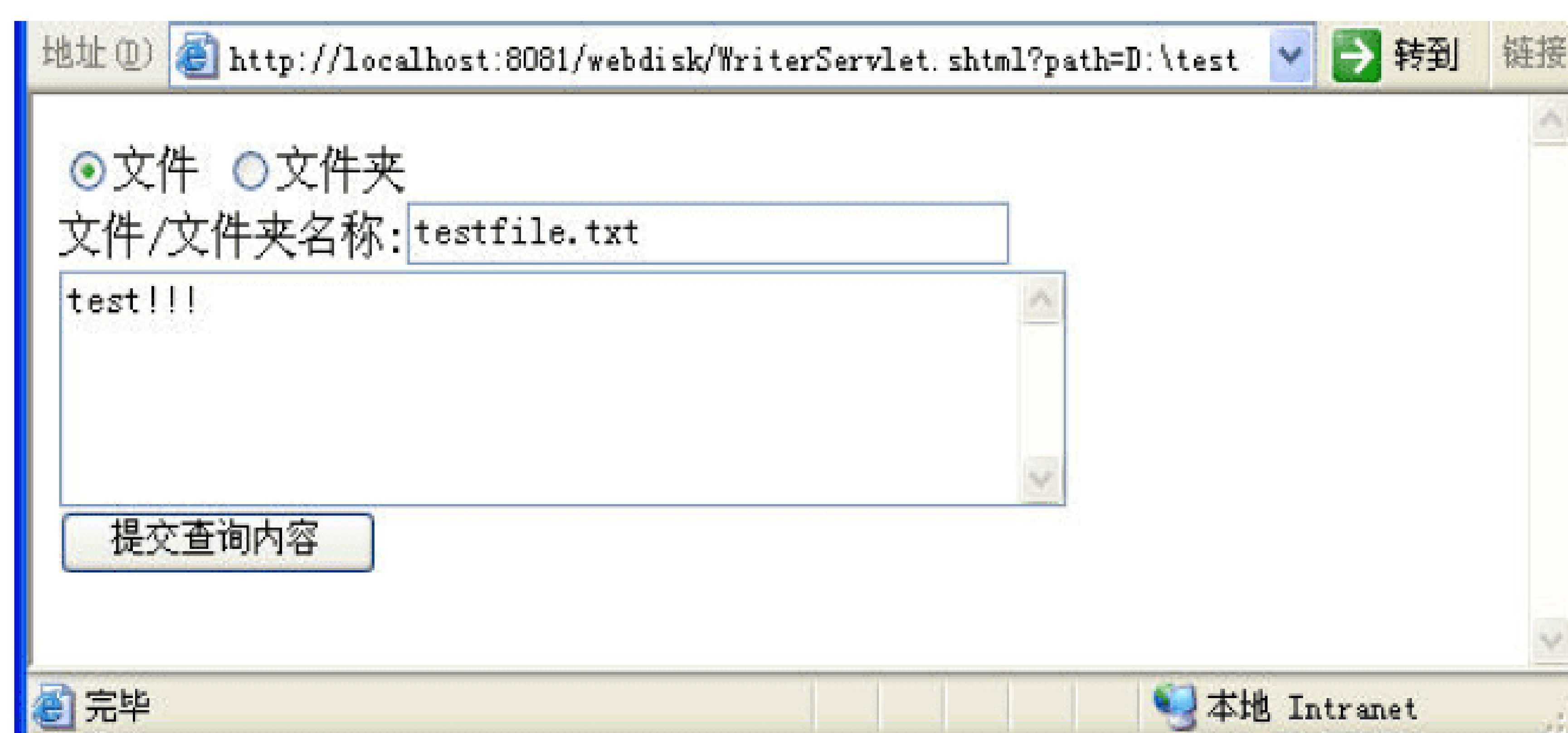


图 6.14　testfile 文件内容

5．操作文件或文件夹

当单击列表中的“删除”链接时，就可以实现对该列表选项的删除功能。例如删除名为 test.txt 的文件时，可以单击图 6.12 中 test.txt 对应的“删除”链接就可以转到显示 D 盘列表的页面（如图 6.15 所示），该页面与操作前的页面相比则没有名为 test.txt 的列表目录。当删除文件夹时，则必须先删除该文件夹下的文件。例如如果想删除 test 文件夹，则必须先删除该文件夹下名为 testfile.txt 的文件，然后单击 test 文件夹对应的“删除”链接则会实

现该功能，最后页面如图 6.16 所示。



图 6.15　删除 test.txt 文件后页面

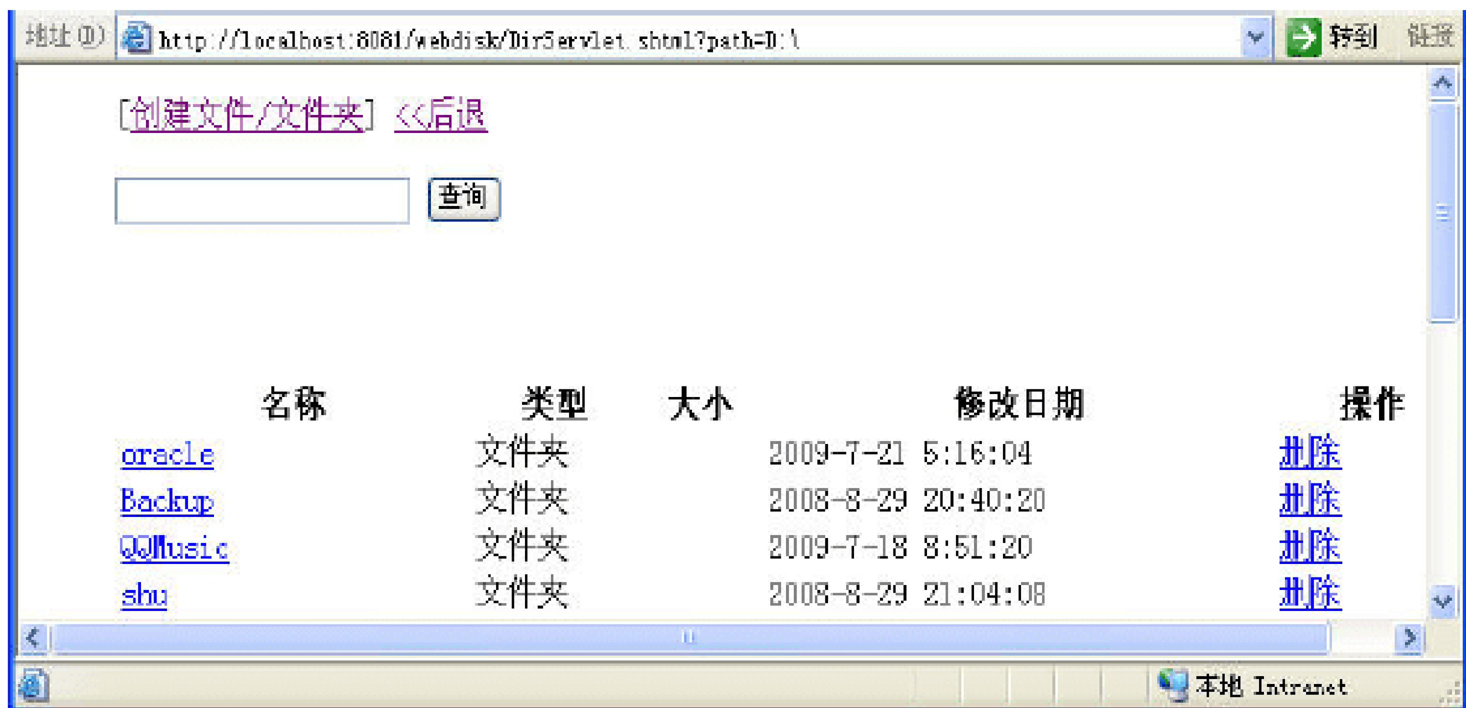


图 6.16　删除 test 文件夹后的页面

6.2　网络硬盘功能具体实现——浏览磁盘和显示文件信息

本节通过 JSP+Servlet 技术来实现网络硬盘功能，在该项目中包含了两种类型的 Servlet 程序，一类为工具类：FileUtils.java 和 StringUtils.java，这些类用来被实现各种功能的 Servlet 程序调用；而另一类则为实现各种功能的 Servlet 类。

本章除了介绍工具类外，还将详细介绍如何浏览磁盘、如何浏览磁盘中的文件、如何显示文件夹和文件信息，具体流程如图 6.17 所示。

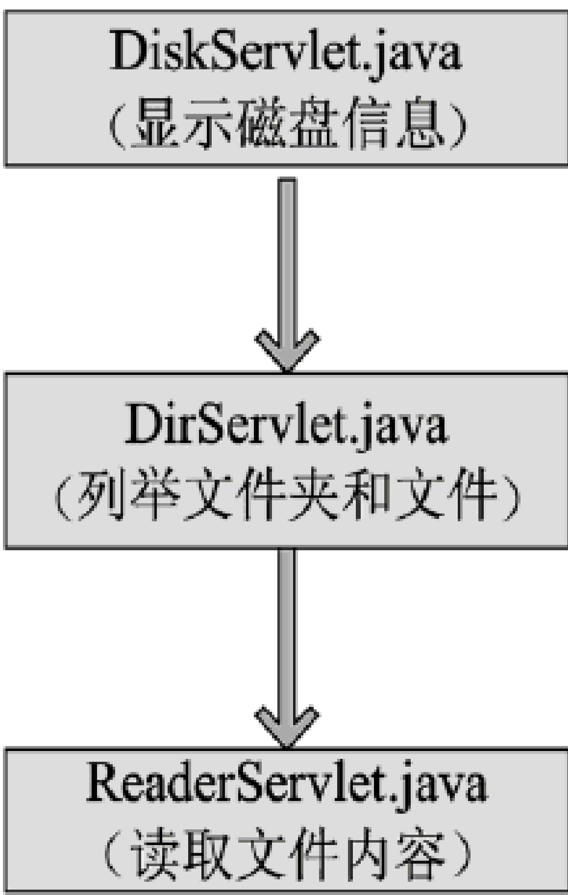


图 6.17　流程图

6.2.1　实现相关工具类

在实现网络硬盘功能的系统中经常会遇到关于文件的各种操作：创建文件、删除文件、读取文件等。为了提高程序的阅读性、维护性，在该系统中创建了一个名为 FileUtils.java 类来实现对文件的各种操作，该类的具体内容如代码 6.1 所示。

代码 6.1　实现文件操作工具类：FileUtils.java

```
...
public class FileUtils {
    public static File[] fileList(String path) {
```

```
                                   //实现得到路径下对应所有的文件及文件夹
        File file = new File(path);
        return file.listFiles();
    }
    //实现获取得到路径下符合条件的文件及文件夹
    public static File[] fileList(String path, final String cond) {
        File file = new File(path);
        return file.listFiles(new FilenameFilter() {
            public boolean accept(File dir, String name) {
                return name.indexOf(cond) >= 0;
            }
        });
    }
    public static String readFile(String path) throws IOException {
                                   //实现读取文件功能
        File file = new File(path);
        StringBuffer strBuffer = new StringBuffer();
        Reader reader = null;
        BufferedReader breader = null;
        try {
            reader = new FileReader(file);
            breader = new BufferedReader(reader);
            String strline = null;
            while ((strline = breader.readLine()) != null) {
                strBuffer.append(strline + "\n");
            }
        } catch (FileNotFoundException e) {
            throw new FileNotFoundException("文件不能被发现！");
        } catch (IOException e) {
            throw new IOException("文件读取中出现了异常！", e);
        } finally {
            if (breader != null)
                breader.close();
            if (reader != null)
                reader.close();
        }
        return strBuffer.toString();
    }
   public static void writeFile(String path , String content) throws
   IOException{                    //实现写文件功能
       File file = new File(path);
       Writer writer = null;
       BufferedWriter bwriter = null;
       PrintWriter out = null;
       try {
            writer = new FileWriter(file , true);
            bwriter = new BufferedWriter(writer);
            out = new PrintWriter(bwriter);
            out.println(content);
        } catch (IOException e) {
            throw new IOException("写文件出现了异常！" , e);
        } finally {
            if(out != null) out.close();
            if(bwriter != null) bwriter.close();
            if(writer != null) writer.close();
        }
    }
    public static boolean createFile(String path) throws IOException{
                                   //实现创建文件功能
        File file = new File(path);
```

```
        try {
            return file.createNewFile();
        } catch (IOException e) {
            throw new IOException("文件创建失败！" , e);
        }
    }
    public static boolean mkdirs(String path){      //实现创建多个文件夹功能
        File file = new File(path);
        return file.mkdirs();
    }
    public static boolean delete(String path){      //实现删除文件和文件夹功能
        File file = new File(path);
        return file.delete();
    }
}
```

【代码解析】

在上述代码中，通过 JDK 提供的关于 IO 和 File 的 API 类，实现一些常用的操作文件的功能。各个方法的具体作用如下。

- fileList(path)：获取路径下的所有文件和文件夹。
- fileList(path,cond)：获取路径下符合条件的文件和文件夹。
- readFile()：读取文件。
- writeFile()：写入文件。
- createFile()：创建文件。
- mkdirs()：创建多个文件夹。
- delete()：删除文件功能。

在网络硬盘功能系统中为了提高系统的安全性，创建了一个名为 StringUtils.java 的类过滤掉一些特殊的字符，该类的具体内容如代码 6.2 所示。

代码 6.2　过滤特殊符号工具类：StringUtils.java

```
...
    public static String filterHTML(String str){
        StringBuffer strs = new StringBuffer();
        if(str != null && !str.equals("")){
            char[] chs = str.toCharArray();
            for(char c : chs){
            switch(c){
                case ' ' : strs.append(" "); break;
                case '>' : strs.append("&gt;"); break;
                case '<' : strs.append("&lt;"); break;
                case '\n' : strs.append("<br>"); break;
                default : strs.append(c);
                }
            }
        }
        return strs.toString();
    }
}
```

【代码解析】

在该系统中当实现显示文件内容功能时，会遇到特殊字符，这时就需要通过调用该类来过滤掉这些特殊字符。

注意：为了提高系统的安全，是不允许上传各种代码。为了实现该功能，就需要上述的过滤特殊符号工具类。

6.2.2　浏览磁盘

在网络硬盘功能系统中，首先需要浏览服务器端的磁盘，然后才能在相应的磁盘中创建、删除或查找相关的文件或文件夹。在网络硬盘功能的系统中通过 DiskServlet.java 类来实现浏览磁盘功能，该类的具体内容如代码 6.3 所示。

代码 6.3　显示磁盘信息：DiskServlet.java

```
...
public class DiskServlet extends HttpServlet {
    //编写 doGet()方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
        PrintWriter out = response.getWriter();
        //输出页面的相关内容
...
        out.println("  <BODY>");
        File[] roots = File.listRoots();    //获取根目录下的磁盘
        for(File f : roots){                //遍历对象 roots
            out.println("<a href='DirServlet.shtml?path=" + f.
            getAbsolutePath() + "'>服务器磁盘:(" + f.getAbsolutePath()+")
            </a>   ");
        }
        out.println("  </BODY>");
...
        out.flush();
        out.close();
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        this. doGet(reques, response);      //调用 doGet()方法
    }
}
```

接着在 web.xml 文件中实现对该 Servlet 程序的配置。

```
<!--配置 DiskServlet 类路径-->
<servlet>
  <servlet-name>DiskServlet</servlet-name>
  <servlet-class>com.cjg.web.DiskServlet</servlet-class>
</servlet>
<!--配置 DiskServlet 类路径-->
<servlet-mapping>
  <servlet-name>DiskServlet</servlet-name>
  <url-pattern>/DiskServlet.shtml</url-pattern>
</servlet-mapping>
```

🔔注意：在上述代码中，首先通过调用类 File 获取根目录下的所有磁盘，然后通过遍历所有磁盘输出各个磁盘的名称。

6.2.3　浏览磁盘中的文件夹和文件

如果想查看磁盘中文件夹和文件的信息，就需要实现浏览磁盘中的文件夹和文件功能。在网络硬盘功能的系统中通过 DirServlet.java 类来实现浏览文件夹和文件的功能，该类的具体内容如代码 6.4 所示。

代码 6.4　显示文件夹和文件：DirServlet.java

```
...
public class DirServlet extends HttpServlet {
    //编写 doGet()方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
        PrintWriter out = response.getWriter();
        //输出页面的相关内容
...
        out.println("  <BODY leftmargin=50>");
        String path = request.getParameter("path");
        if( path != null && !path.equals("")){
            out.println("[<a href='WriterServlet.shtml?path=" + path + "'>
            创建文件/文件夹</a>]");
            File file = new File(path);
            out.println("<a href='DirServlet.shtml?path=" + file.getParent
            () + "'><<后退</a>");
            out.println("    ");
            out.println("<form action='QueryServlet.shtml' method=
            post>");
            out.println("<input type=hidden name=path value='" + path +
            "'>");
            out.println("<input name=query> <input type=submit value=查
            询>");
            out.println("</form>");
            //得到根目录下的文件夹及文件
            File[] files = FileUtils.fileList(path);
            //过滤隐藏文件，并展现
            out.println("<table align=left border=0 width=700>");
            out.println("<tr>");
            out.println("<th>名称</th><th>类型</th><th>大小</th><th>修改日
            th><th>操作</th>");
            for(File f : files){
                if(!f.isHidden()){
                    out.println("<tr>");
                    String url = f.isDirectory() ? "DirServlet.shtml?path="
                    +f.getAbsolutePath() : "ReaderServlet.shtml?
                    path=" + f.getAbsolutePath();
                    out.print("<td><a href='" + url + "'>" + f.getName() +
                    "</a></td>");
                    out.print("<td>" + (f.isFile() ? "文件" : "文件夹") +
```

```
                    "</td>");
                    out.println("<td>" + (f.isDirectory() ? "" : (f.length()
                    / 1024) + "KB" ) + "</td>");
                    out.println("<td>" + new Date(f.lastModified()).
                    toLocaleString() + "</td>");
                    out.print("<td><a href='OperateServlet.shtml?mtype=
                    delete&path=" + f.getAbsolutePath() + "'>删除
                    </a>   ");
                    out.println("</tr>");
                }
            }
            out.println("</table>");
        } else {
            response.sendRedirect("DiskServlet.shtml");  //重新得到根目录
        }
        out.println("  </BODY>");
...
        out.flush();
        out.close();
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        this. doGet(reques, response);                   //调用 doGet()方法
    }
}
```

接着在 web.xml 文件中实现对该 Servlet 程序的配置。

```
<!--配置 DirServlet 类路径-->
<servlet>
  <servlet-name>DirServlet</servlet-name>
  <servlet-class>com.cjg.web.DirServlet</servlet-class>
</servlet>
<!--配置 DirServlet 映射路径-->
<servlet-mapping>
  <servlet-name>DirServlet</servlet-name>
  <url-pattern>/DirServlet.shtml</url-pattern>
</servlet-mapping>
```

【代码解析】

- 在上述系统中首先通过调用工具类 FileUtils 中的 fileList()方法来获取该磁盘中的所有文件夹和文件。
- 接着遍历获取的文件夹和文件，在遍历的具体过程中，通过 File 类中的 getName()方法获取该对象的名称；通过 File 类中的 isFile()方法判断该对象是文件还是文件夹；通过 File 类中的 length()方法获取该对象的大小；通过 File 类中的 lastModified()方法获取该对象的最后修改时间。

注意：在上述代码中，主要是通过 File 类的各种方法来显示文件夹和文件的功能。

6.2.4　显示文件夹和文件的内容

当磁盘中的对象为文件夹时，单击该文件夹链接显示文件夹中的文件；当磁盘中的对

象为文件时，单击该链接显示该文件的内容。如何显示文件夹和文件的内容呢？在网络硬盘功能系统中通过名为 ReaderServlet.java 的文件来实现显示文件夹和文件的内容，该类的具体内容如代码 6.5 所示。

代码 6.5　读取文件内容类：ReaderServlet.java

```
...
public class ReaderServlet extends HttpServlet {
    // 编写 doGet()方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
        PrintWriter out = response.getWriter();
        //输出页面的相关内容
...
        out.println("  <BODY>");
        String path = request.getParameter("path");
        if(path != null && !path.equals("")){
            //将请求的文件内容在网页中展现
            String content = FileUtils.readFile(path);
           out.println(StringUtils.filterHTML(content));
        } else {
            response.sendRedirect("DiskServlet.shtml");
        }
        out.println("  </BODY>");
...
        out.flush();
        out.close();
    }
    // 编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        this. doGet();
    }
}
```

接着在 web.xml 文件中实现对该 Servlet 程序的配置。

```
<!--配置 ReaderServlet 类路径-->
<servlet>
  <servlet-name>ReaderServlet</servlet-name>
  <servlet-class>com.cjg.web.ReaderServlet</servlet-class>
</servlet>
<!--配置 ReaderServlet 映射路径-->
<servlet-mapping>
  <servlet-name>ReaderServlet</servlet-name>
  <url-pattern>/ReaderServlet.shtml</url-pattern>
</servlet-mapping>
```

注意： 在上述代码中，通过调用工具类 FileUtils 中的 readFile()方法读取相应文件的内容。

6.3　网络硬盘功能具体实现——操作文件夹和文件

本节通过 Servlet 技术来实现网络硬盘功能，在该项目中包含了两种类型的 Servlet 程序，一类为工具类：FileUtils.java 和 StringUtils.java，而另一类则为实现各种功能的 Servlet 类。

本章将详细介绍如何创建文件夹和文件、删除文件夹和文件，以及查找文件夹和文件，具体流程如图 6.18 所示。

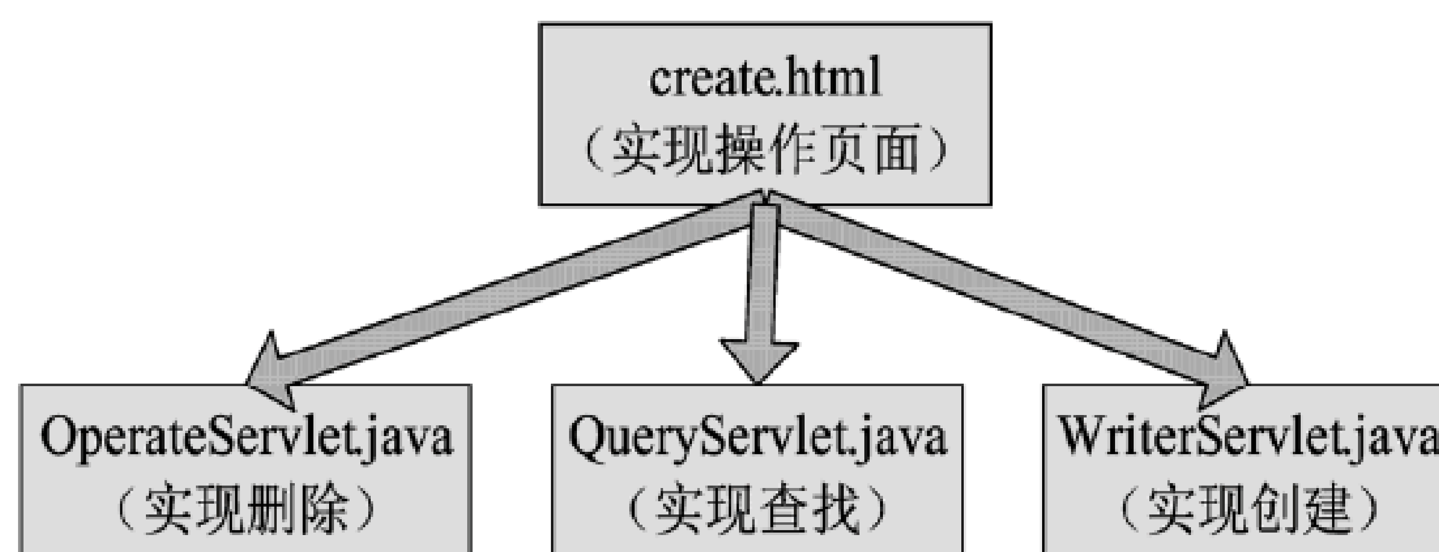


图 6.18　项目流程

6.3.1　删除文件夹和文件

在网络硬盘功能的系统中通过 OperateServlet.java 类来实现删除文件夹和文件的功能，该类的具体内容如代码 6.6 所示。

代码 6.6　删除文件夹和文件：OperateServlet.java

```
...
public class OperateServlet extends HttpServlet {
    //编写 doGet 方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
        //获取传递参数
        String mtype = request.getParameter("mtype");
        String path = request.getParameter("path");
        if(mtype != null && path != null){              //判断参数 path
            if("delete".equals(mtype)){
            FileUtils.delete(path);                      //实现删除功能
        }
        response.sendRedirect("DirServlet.shtml?path="+path.substring(0,
        path.lastIndexOf("\\") + 1));
       }
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        this.doGet();                                    //调用 doGet()方法
    }
}
```

接着在 web.xml 文件中实现对该 Servlet 程序的配置。

```
<!--配置 OperateServlet 类路径-->
<servlet>
  <servlet-name>OperateServlet</servlet-name>
  <servlet-class>com.cjg.web.OperateServlet</servlet-class>
</servlet>
<!--配置 OperateServlet 映射路径-->
<servlet-mapping>
  <servlet-name>OperateServlet</servlet-name>
  <url-pattern>/OperateServlet.shtml</url-pattern>
</servlet-mapping>
```

注意：在上述代码中首先获取传递过来的查询参数，然后通过工具类 FileUtils 的 delete()方法来实现删除文件和文件夹功能。

6.3.2　查找文件夹和文件

在网络硬盘功能的系统中通过 QueryServlet.java 类来实现查找文件夹和文件的功能，该类的具体内容如代码 6.7 所示。

代码 6.7　查找文件夹和文件：QueryServlet.java

```
...
public class QueryServlet extends HttpServlet {
    //编写 doGet()方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        this.doPost();
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
        PrintWriter out = response.getWriter();
        //获取传递参数
        String path = request.getParameter("path");
        String query = request.getParameter("query");
        //输出页面的相关内容
...
        out.println("  <BODY leftmargin=50>");
        if(query.equals("")){
            out.println("请选择需要查找的内容！");
        } else {
            File[] files = FileUtils.fileList(path, query);  //实现查找功能
            //过滤隐藏文件，并展现
            out.println("<table align=left border=0 width=700>");
...
            for(File f : files){                              //遍历文件
                if(!f.isHidden()){
                    out.println("<tr>");
```

```
                    String url = f.isDirectory() ? "DirServlet.shtml?path="
                    + f.getAbsolutePath() : "ReaderServlet.shtml?
                    path=" + f.getAbsolutePath();
                    out.print("<td><a href='" + url + "'>" + f.getName() +
                    "</a></td>");
                    out.print("<td>" + (f.isFile() ? "文件" : "文件夹") +
                    "</td>");
                    out.println("<td>" + (f.isDirectory() ? "" : (f.length()
                    / 1024) + "KB" ) + "</td>");
                    out.println("<td>" + new Date(f.lastModified()).
                    toLocaleString() + "</td>");
                    out.print("<td><a href='OperateServlet.shtml?
                    mtype=delete&path=" + f.getAbsolutePath() + "'>删除
                    </a>   ");
                    out.print("<a href='OperateServlet.shtml?
                    mtype=rename&path=" + f.getAbsolutePath() + "'>更名
                    </a></td>");
                   out.println("</tr>");
                }
            }
            out.println("</table>");
        }
        out.println("  </BODY>");
...
        out.flush();
        out.close();
    }
}
```

接着在 web.xml 文件中实现对该 Servlet 程序的配置。

```
<!--配置 QueryServlet 类路径-->
<servlet>
  <servlet-name>QueryServlet</servlet-name>
  <servlet-class>com.cjg.web.QueryServlet</servlet-class>
</servlet>
<!--配置 QueryServlet 映射路径-->
<servlet-mapping>
  <servlet-name>QueryServlet</servlet-name>
  <url-pattern>/QueryServlet.shtml</url-pattern>
</servlet-mapping>
```

注意: 在上述代码中首先获取传递过来的查询参数，然后通过工具类 FileUtils 的 fileList() 方法来实现查找文件和文件夹的功能。

6.3.3　创建文件夹和文件

在网络硬盘功能的系统中，通过 WriterServlet.java 类来实现创建文件夹和文件的功能，该类的具体内容如代码 6.8 所示。

代码 6.8　创建文件和文件夹类：WriterServlet.java

```
...
public class WriterServlet extends HttpServlet {
    //编写 doGet()方法
```

```
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
        PrintWriter out = response.getWriter();
        //输出页面的相关内容
...
        out.println("  <BODY>");
        String path = request.getParameter("path");          //获取 path 参数
        if(path != null && !path.equals("")){                //判断 path 参数
        out.println("<form action='WriterServlet.shtml' method=post>");
        out.println("<input type=hidden name=path value='" + path + "'>");
        out.println("<input name=type type=radio value=1>文件");
        out.println("<input name=type type=radio value=2>文件夹");
        out.println("<br>文件/文件夹名称:<input name=name size=30>");
        out.println("<br>");
        out.println("<textarea rows=5 cols=50 name=content></textarea>
        <br>");
        out.println("<input type=submit>");                  //提交按钮
        out.println("</form>");
        out.println("  </BODY>");
...
        out.flush();
        out.close();
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
        PrintWriter out = response.getWriter();
        //输出页面的相关内容
...
        out.println("  <BODY>");
        //获取参数
        String path = request.getParameter("path");
        String name = request.getParameter("name");
        String type = request.getParameter("type");
        if(type.equals("1")){                                //创建文件
            String content = request.getParameter("content");
            FileUtils.writeFile(path + "\\" + name, content);
        } else {                                             //创建文件夹
            boolean result = FileUtils.mkdirs(path + "\\" + name);
           if(!result){
            out.println("创建失败了！");
            out.println("<a href='#'>返回</a>");
           }
        }
        response.sendRedirect("DirServlet.shtml?path=" + path);
        out.println("  </BODY>");
...
        out.flush();
        out.close();
    }
}
```

接着在 web.xml 文件中实现对该 Servlet 程序的配置。

```
<!--配置 WriterServlet 类路径-->
<servlet>
  <servlet-name>WriterServlet</servlet-name>
  <servlet-class>com.cjg.web.WriterServlet</servlet-class>
</servlet>
<!--配置 WriterServlet 映射路径-->
<servlet-mapping>
  <servlet-name>WriterServlet</servlet-name>
  <url-pattern>/WriterServlet.shtml</url-pattern>
</servlet-mapping>
```

【代码解析】

在上述代码中首先获取传递过来的参数，然后通过工具类 FileUtils 的 mkdirs()创建文件夹，通过工具类 FileUtils 的 writeFile()创建文件。

6.4　小　　结

本章主要介绍网络磁盘功能，该功能基于 JSP+Servlet 解决方案。网络磁盘功能在业务上主要分成两部分：列举磁盘和显示文件信息、操作文件和文件夹。在具体实现列举目录功能时，不仅实现了列举磁盘信息、文件夹信息和文件信息的功能，而且还实现了显示文件内容的功能。在具体操作文件和文件夹时，不仅实现了查找文件和文件夹的功能，而且还实现了创建、删除文件和文件夹的功能。

第 7 章　网站统计模块（JSP+Servlet）

任何网络系统都离不开统计模块，因为这些统计结果可以显示出该网站的吸引力或者网站程序的运行效率。在开发具体网站时一般都会加入一些必要的统计功能：用来描述网站拥有用户数量的统计用户人数功能；用来描述网站流量的统计在线人数功能。同时为了使网站更具有人性化，可以在浏览者在登录后，在浏览的网页上显示欢迎该浏览者的信息。

本章将通过 JSP+Servlet 框架技术来介绍如何实现这些统计模块：

- 显示浏览者信息；
- 统计访问量；
- 统计在线人数。

7.1　网站统计模块原理

网站的统计模块表面看来很复杂，其实实现起来很简单，只需要使用 JSP 语言中的一些对象就可以。只要浏览者登录成功后，就会在所有请求的页面上显示出统计的相关结果。

7.1.1　网站统计模块框架分析

对于一个要求比较严格的网上系统来说，实现可用的网站统计功能需要考虑许多的细节问题，例如如何在显示统计信息时更人性化等。该章将会实现比较简单可用的网站统计功能，读者可以根据自己的需求自行进行完善。本系统的结构框架如图 7.1 所示。

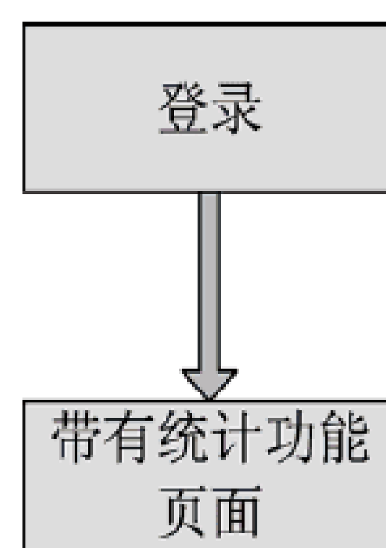


图 7.1　系统流程图

7.1.2　网站统计功能描述

本节将以直观的方式向读者介绍网站统计模块要实现的功能。这些功能包括显示浏览者信息、统计访问量和统计在线人数。

1．显示浏览者信息

浏览者首先通过浏览登录网页来实现登录，如图 7.2 所示。

当单击 submit 按钮后，用户信息就会成功写入 Cookie 中。这时页面就会转到设置 Cookies 页面，如图 7.3 所示。这时单击“查看 Cookies”链接就可以转到如图 7.4 所示的显示浏览者信息的页面。

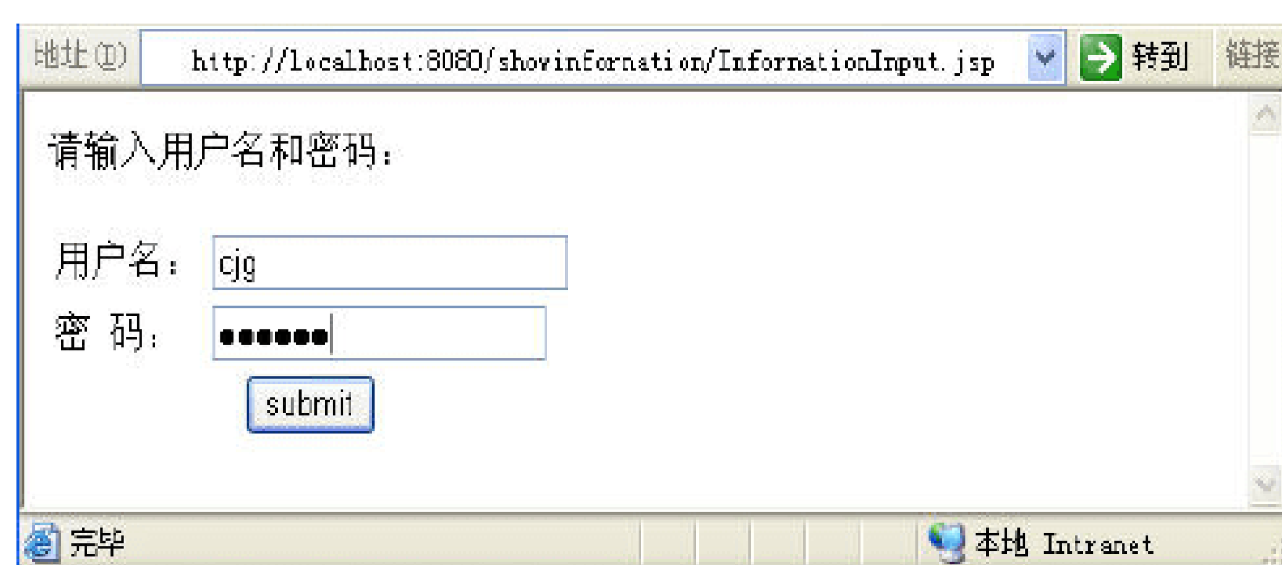
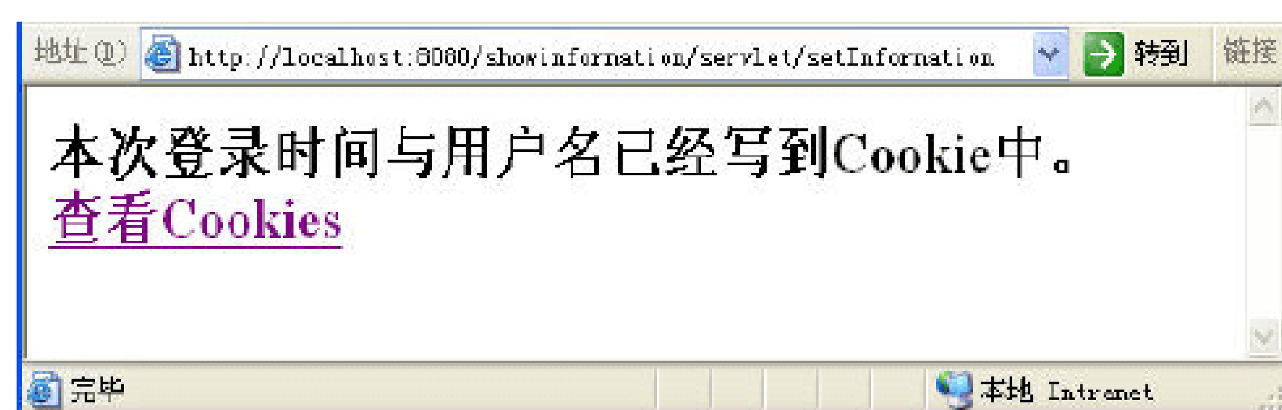


图 7.2　登录



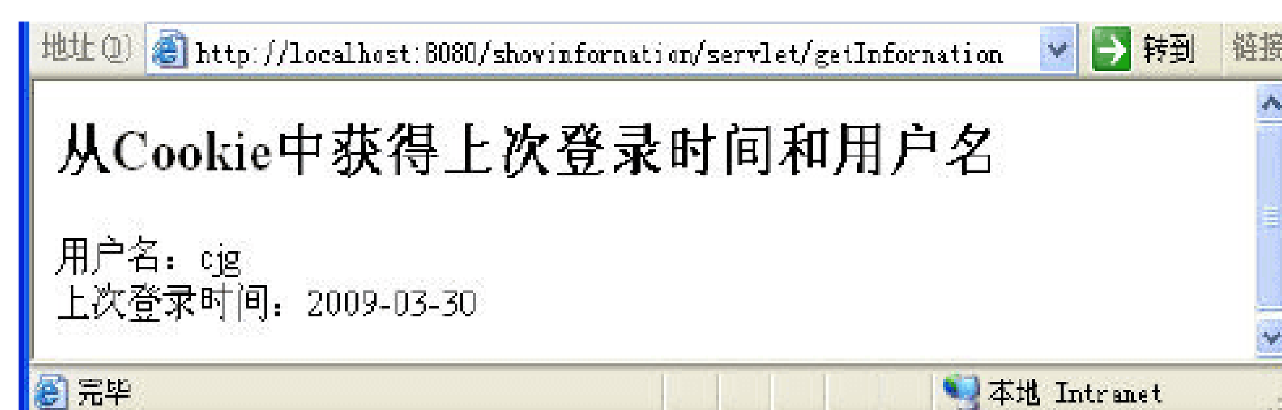
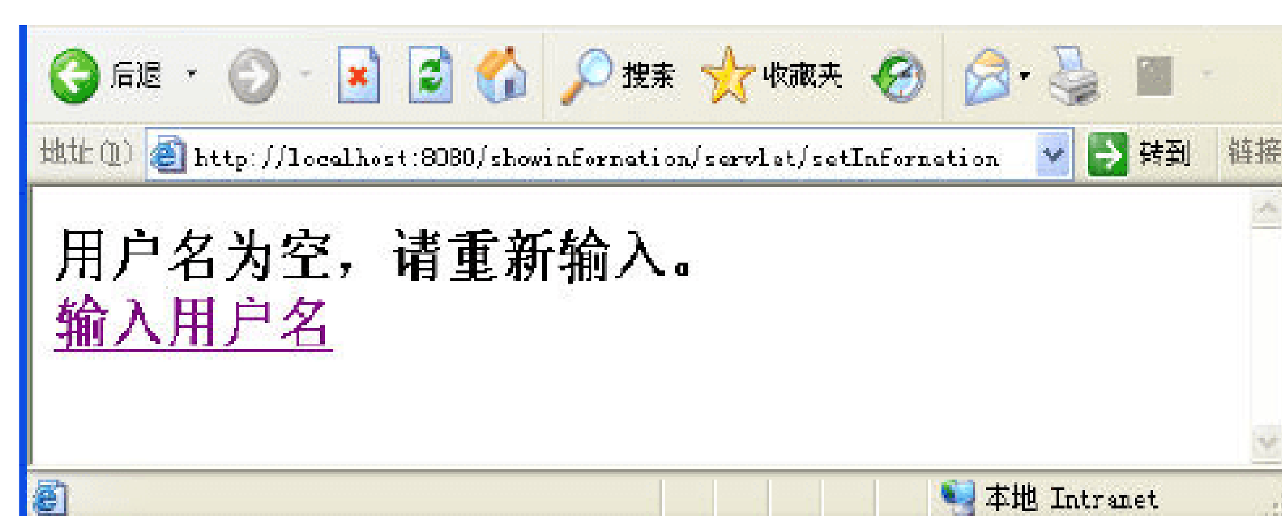


图 7.3　成功写入 Cookie　　　　图 7.4　显示浏览者信息

当用户信息没成功地写到 Cookie 后，就会转到重新输入页面，如图 7.5 所示。这时单击“输入用户名”链接就可以转到如图 7.6 所示的登录页面。



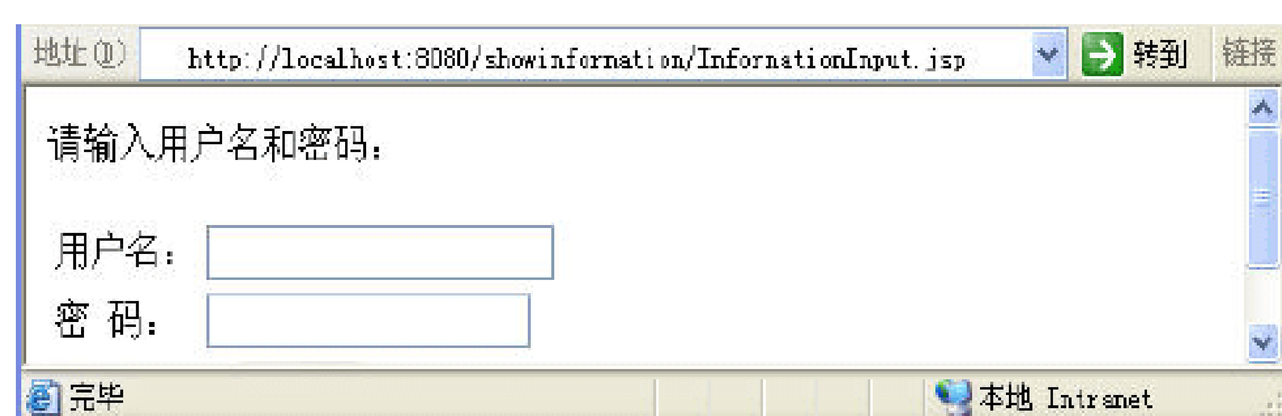
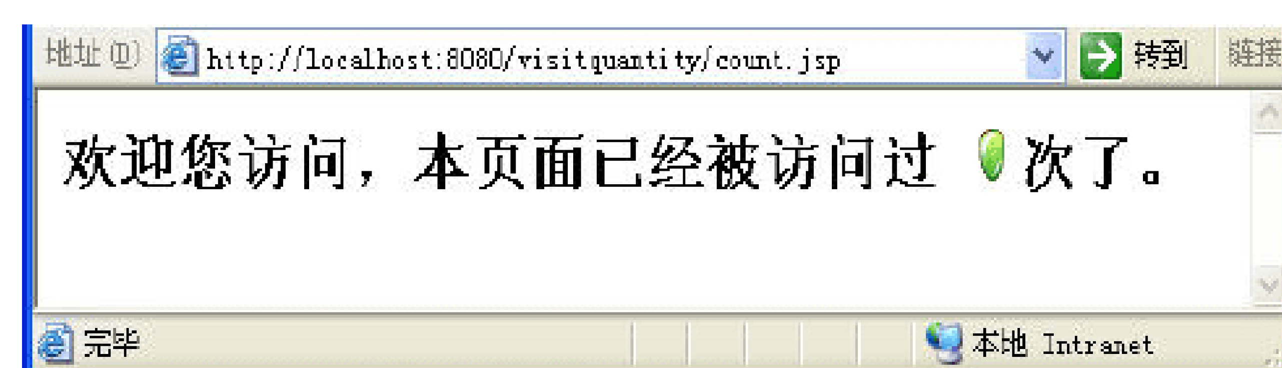


图 7.5　重新输入页面　　　　图 7.6　登录页面

2．统计访问量

当浏览者访问 count.jsp 页面时，就会出现如图 7.7 所示的显示访问量的页面。该浏览者在打开 count.jsp 页面后，连续地单击“刷新”按钮，就会使访问量每刷新一下加 1，如图 7.8 所示。



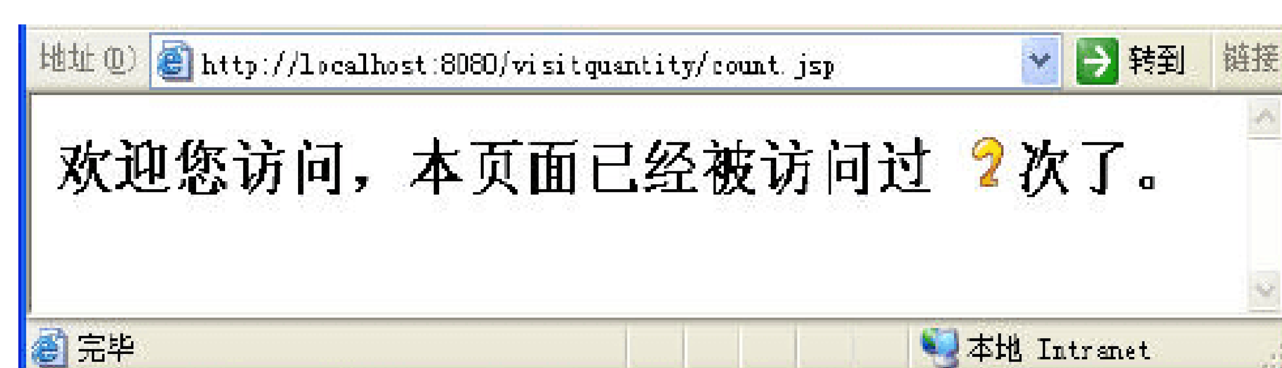


图 7.7　统计访问量页面　　　　图 7.8　刷新后统计访问量页面

当浏览者访问 count2.jsp 页面时，也会出现如图 7.9 所示的显示访问量的页面。但是当该浏览者在打开 count2.jsp 页面后，连续地单击“刷新”按钮，访问量不会每刷新一下访问就增加 1，还是如图 7.9 所示。

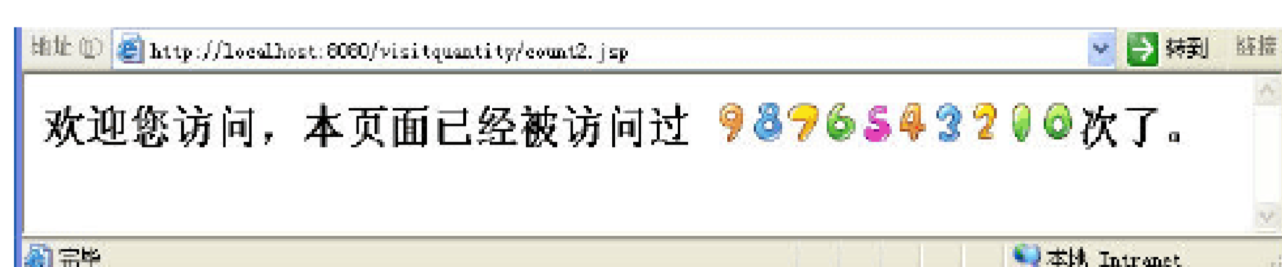


图 7.9　限制刷新的统计访问量页面

注意： 之所以会出现图 7.9 那么大的访问次数，其实不需要真实的访问这么多次，只需要在该项目中修改存放数字文件（Tomcat 6.0\webapps\visitquantity\count.txt）中的数字就可以。

3．统计在线人数

当浏览者连续访问同时不关闭 showpeople.jsp 页面时，就会出现如图 7.10 所示的显示统计在线人数的 3 个页面。3 个页面出现的结果之所以不一样是因为前两个页面没有刷新，对前两个页面分别刷新就会出现如图 7.11 所示的页面。



图 7.10　统计在线人数页面

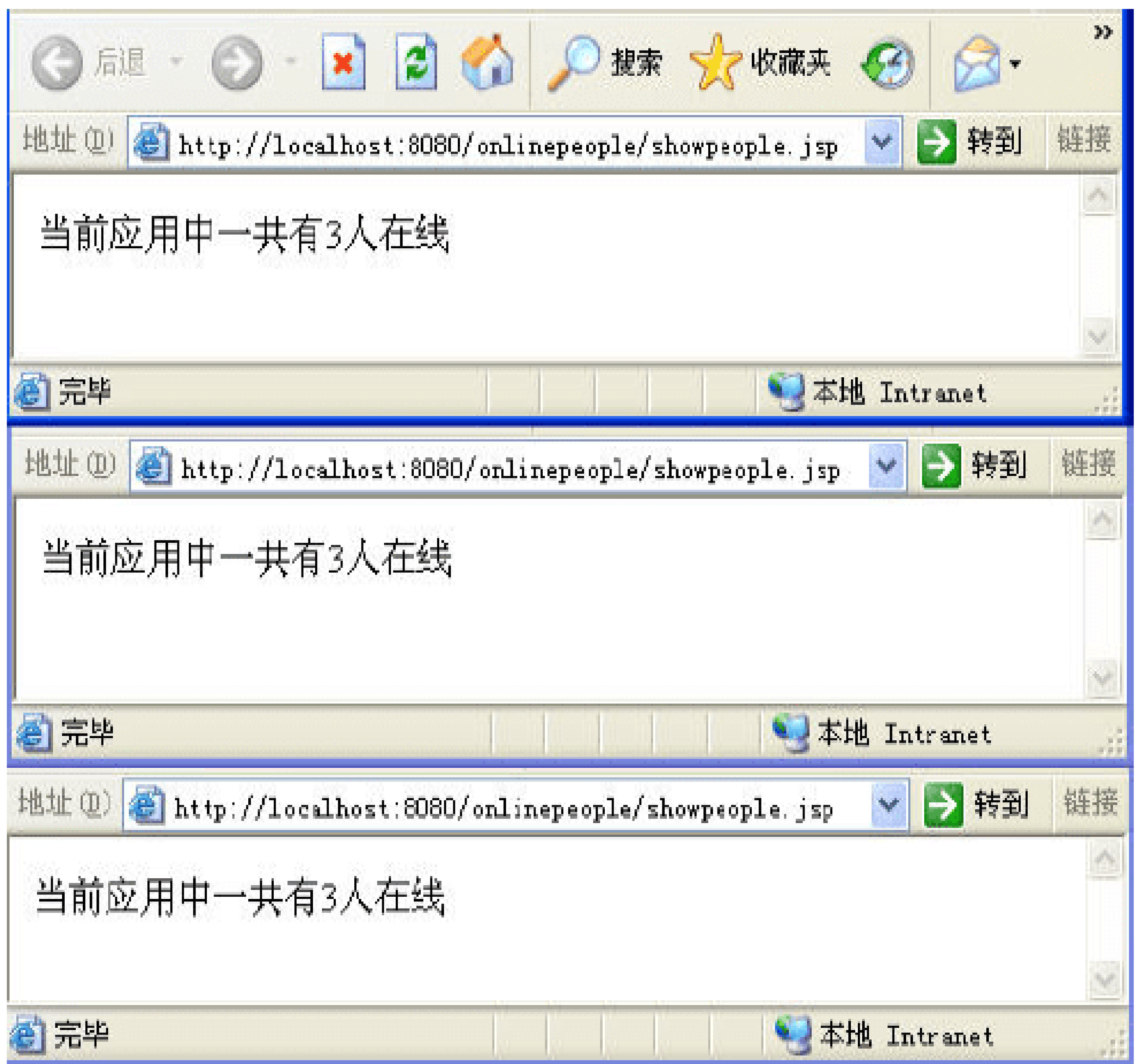


图 7.11　刷新后统计在线人数页面

7.2　实现显示欢迎信息功能

本章通过 JSP+Servlet 框架技术来实现显示欢迎信息，显示欢迎信息程序架构如图 7.12 所示，它包含一个 JSP 页面和两个 Servlet 程序：InfornationInput.jsp、SetInfornationServlet.java 和 GetInfornationServlet.java。

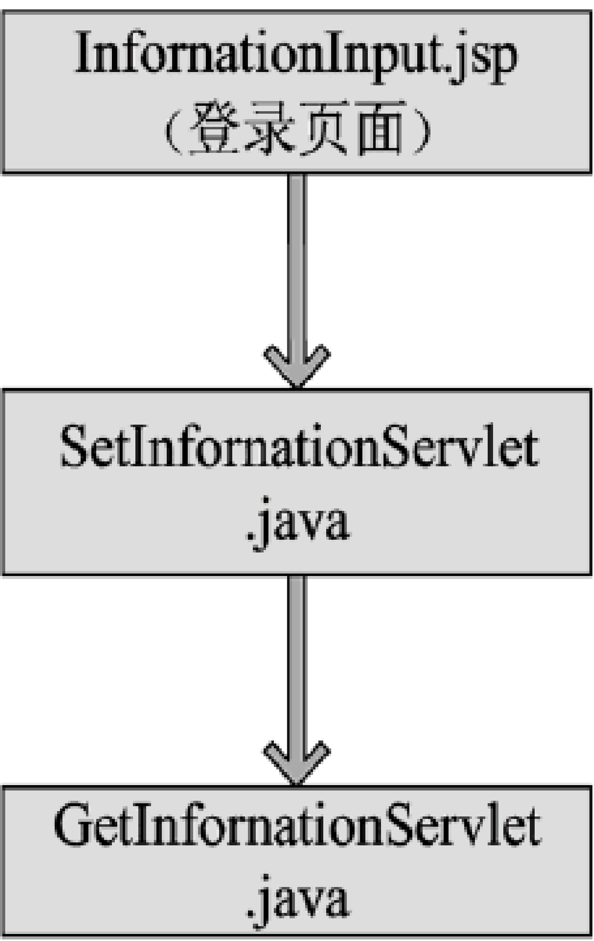


图 7.12　程序关系图

7.2.1　登录页面

InfornationInput.jsp 用来实现浏览者登录功能。当浏览者登录后，就会在页面上显示出浏览者信息。代码 7.1 用来设计登录界面。

代码 7.1　登录页面：InfornationInput.jsp

```
...
<body>
请输入用户名（英文或者数字）
```

```
<!--表单内容-->
<form name="form1" action="/showinfornation/servlet/setInfornation"
method="post" >
用户名：<input type="text" name="username">
密　码：<input type="password" name="userpassword">
<input name="submit" value="submit" type="submit">
</form>
</body>
...
```

【代码解析】

上述代码只是一个普通的登录界面，在该登录页面中只设计了两个文本框和提交按钮。该页面发出的请求交由/showinfornation/servlet/setInfornation 处理。

7.2.2　设置用户信息

SetInfornationServlet 这个服务器端程序用来把用户信息存储到计算机的 Cookie 中，这样就可以在其他页面中获取用户信息。代码 7.2 演示了如何把用户信息存储到本地计算机中。

代码 7.2　设置用户信息：SetInfornationServlet.java

```
...
public class SetCookiesServlet extends HttpServlet {
    //编写 doGet()方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        doPost(request, response);
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        String output = null;                    //定义输出流变量
        String username = request.getParameter("username");
                                                 //定义用户变量存储获取的参数
        String password=request.getParameter("userpassword");
                                                 //获取密码参数
        if (!StringTool.validateNull(username)) {  //判断用户名
            //创建名为 username 的 Cookie
            Cookie cookie1 = new Cookie("username", StringUtil.filterHtml
            (username));
            //cookie 的有效期为 1 个月
            cookie1.setMaxAge(24 * 70 * 70 * 30);
            //设置时间格式
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            //创建名为 lastTime 的 Cookie
            Cookie cookie2 = new Cookie("lastTime", sdf.format(new
            Date()));
            //Cookie 的有效期为 1 个月
            cookie2.setMaxAge(24 * 70 * 70 * 30);
            response.addCookie(cookie1);         //添加 Cookie1 到响应中
            response.addCookie(cookie2);         //添加 Cookie2 到响应中
            //输出内容
```

```
            output = "本次登录时间与用户名已经写到 Cookie 中。<br><a href=\"
            /showinfornation/servlet/getInfornation\ ">查看 Cookies</a>";
                                                    //设置成功时 output 变量值
        } else {
            output = "用户名为空，请重新输入。<br><a href=\" /showinfornation/
            InfornationInput.htm\">输入用户名</a>";
                                                    //设置失败时 output 变量值
        }
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();    //设置输出流
        out.println("<html>");
        out.println("<head><title>set cookies </title></head>");
        out.println("<body>");
        out.println("<h2>" + output + "</h2>");    //输出 output 值
        out.println("</body>");
        out.println("</html>");
        out.flush();
        out.close();
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 SetCookiesServlet 类路径-->
<servlet>
  <servlet-name>SetCookiesServlet</servlet-name>
  <servlet-class>com.cjg.servlet.SetInfornationServlet</servlet-class>
</servlet>
<!--配置 SetCookiesServlet 映射路径-->
<servlet-mapping>
  <servlet-name>SetCookiesServlet</servlet-name>
  <url-pattern>/servlet/setInfornation</url-pattern>
</servlet-mapping>
```

【代码解析】

- 首先定义了 3 个变量 output、username 和 password，第 1 个参数用来表示输出的字符串，第 2 个参数表示用户名，第 3 个参数表示密码。
- 获取传递过来的用户名和密码参数后，首先要对其进行判断。当获取的用户名不为空时，就可以把其存储到 Cookie。由于还需要显示用户登录的时间，所以还需要把登录时间存储到 Cookie 中。最后还需要设置变量 output 的值。
- 最后动态生成一个网页，在该页面中输出 output 的值。

7.2.3　获取用户信息

在动态生成的“设置 Cookies”网页中，当单击“查看”链接时，就会通过 GetInfornationServlet 程序获取存储本地计算机中名为 username 和 lastTime 的 Cookie 中的值，具体内容如代码 7.3 所示。

代码 7.3　获取本地的 Cookie：GetInfornationServlet.java

```
...
public class GetCookiesServlet extends HttpServlet {
    //修改 doGet()方法
```

```
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        doPost(request, response);
    }
    //修改 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();    //设置输出流
        out.println("<html>");
        out.println("<head><title>display login information
        </title></head>");
        out.println("<body>");
        out.println("<h2>从 Cookie 中获得上次登录时间与用户名</h2>");
        Cookie[] cookies = request.getCookies();   //获取 Cookie 数组
        Cookie cookie = null;                      //创建一个 Cookie 对象
        //遍历 Cookie 数组
        for (int i = 0; i < cookies.length; i++) {
            cookie = cookies[i];
            if (cookie.getName().equals("username")) {
                out.println("用户名：" + cookie.getValue());
                                                   //输出 Cookie 对象的值
                out.println("<br>");
            }
            if (cookie.getName().equals("lastTime")) {
                out.println("上次登录时间：" + cookie.getValue());
                                                   //输出 Cookie 对象的值
                out.println("<br>");
            }
        }
        out.println("</body>");
        out.println("</html>");
        out.flush();
        out.close();
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 GetCookiesServlet 类路径-->
<servlet>
  <servlet-name>GetCookiesServlet</servlet-name>
  <servlet-class>com.cjg.servlet.GetInfornationServlet</servlet-class>
</servlet>
<!--配置 GetCookiesServlet 映射路径-->
<servlet-mapping>
<servlet-name>GetCookiesServlet</servlet-name>
  <url-pattern>/servlet/getInfornation</url-pattern>
</servlet-mapping>
```

【代码解析】

- ❑ 上述代码在动态生成网页时，先获取所有 Cookie 存储到数组 Cookie 中，然后遍历该数组 cookies。
- ❑ 在遍历数组 cookies 的过程中，根据 Cookie 的名字进行判断，一当名字为 username 或 lastTime 时，就要输出各自对应的 Cookie 中存储的值。

7.3　指点迷津——Cookie 知识

在 showinfornation 项目中关键技术就是对 Cookie 对象的操作，所谓 Cookie 在英文中是小甜品的意思。之所以会出现在编程语言中，因为在登录网页时会在网页中出现：“你好 XXX”，浏览者看到后就会感到很亲切就像吃了小甜品一样。本节将深入的介绍 Cookie 的相关技术。

7.3.1　Cookie 基础知识

查看相关技术文档，Cookie 是这样定义的：Cookie 是 Web 服务器保存在用户硬盘上的一段文本。Cookie 允许一个 Web 站点在用户的电脑上保存信息并且随后再取回它。信息的内容以“名-值”对（name-valuepairs）的形式储存。

在知道了 Cookie 的具体含义后，那为什么要使用 Cookie 对象呢？要彻底地弄清楚这个问题必须理解无状态的 HTTP 协议。当浏览者在浏览器中输入：www.baidu.com 后，就会发现网址变成：http://www.baidu.com，其原因就是浏览网页是基于 HTTP 协议。之所以会使用 HTTP 协议，因为该协议是无状态——客户端和服务器不需要彼此记录对方，即 HTTP 协议无法记录浏览者的信息。为了弥补 HTTP 协议的缺陷，就需要使用 Cookie。Cookie 在浏览器与 Web 服务器之间的传送过程如图 7.13 所示。

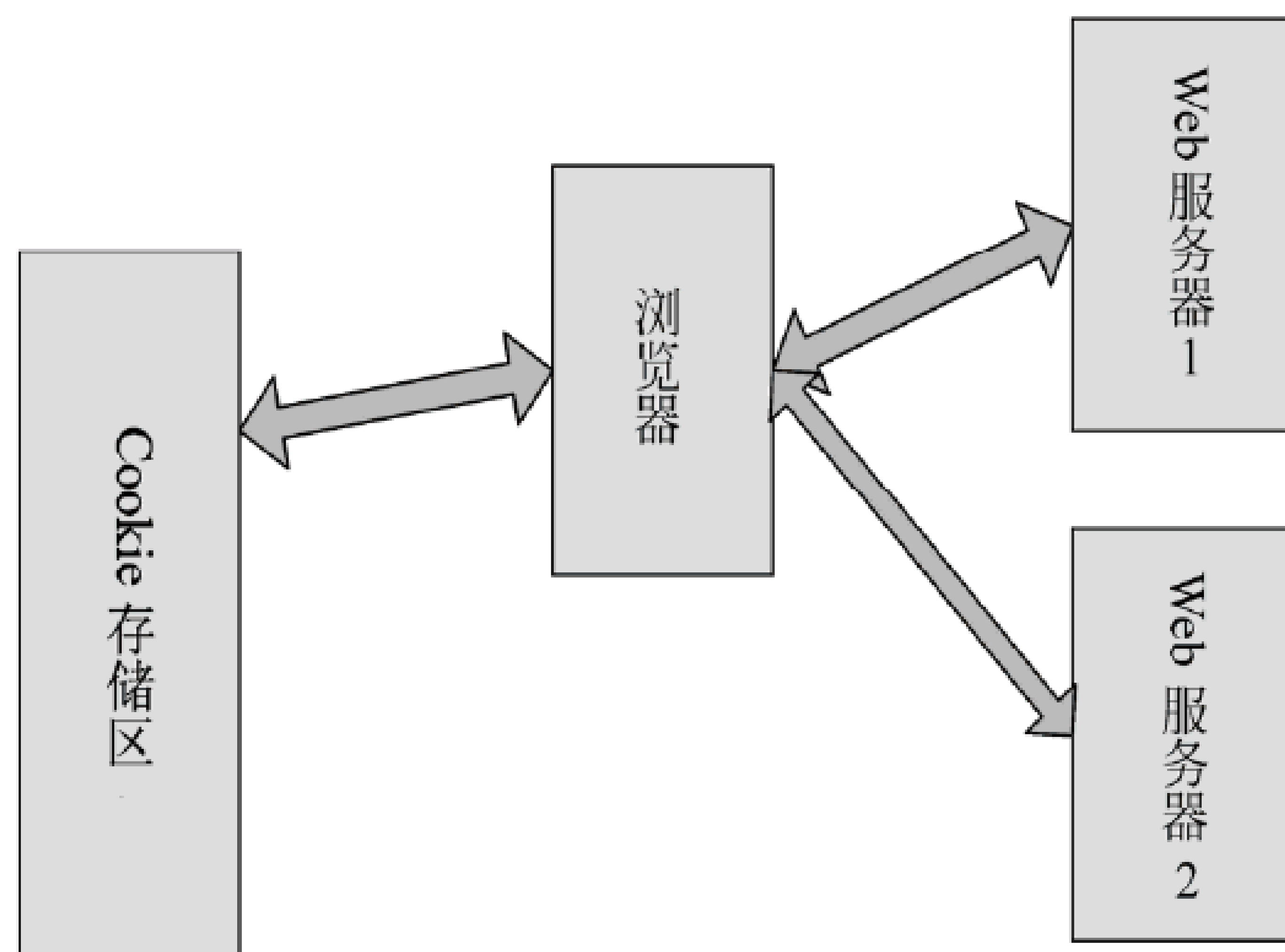


图 7.13　Cookie 传送过程

（1）当浏览器第一次访问服务器的某个资源（Servlet1）时，Web 服务器在 HTTP 响应消息头中附带传送给浏览器的段数据，该数据就会存储在浏览器的 Cookie 中。

（2）当浏览器保存了关于 Servlet1 的 Cookie 后，以后浏览器每次访问该 Web 服务器中的 Servlet1 资源，就会在 HTTP 请求头中将 Cookie 中的数据回传给 Web 服务器。

注意： 服务器决定是否发送 Cookie 和发送 Cookie 的内容，但是浏览器决定是否保存 Cookie。

既然是由浏览器决定存储 Cookie，那么在 IE 浏览器中如何设置 Cookie？首先选择 IE

浏览器的“工具”|Internet 命令，打开如图 7.14 所示的“Internet 选项”对话框。

在“Internet 临时文件”区域中，存在 3 个按钮：删除 Cookies、删除文件和设置按钮。

- 删除 Cookies：删除本地计算机中的 Cookies。
- 删除文件：删除本地计算机中 Internet 临时文件夹中的所有内容。
- 设置：在出现的“设置”对话框（如图 7.15 所示）中可以通过单击 Internet 临时文件夹选项卡中的三个按钮中“查看文件…”按钮，打开存放 Cookies 的文件。

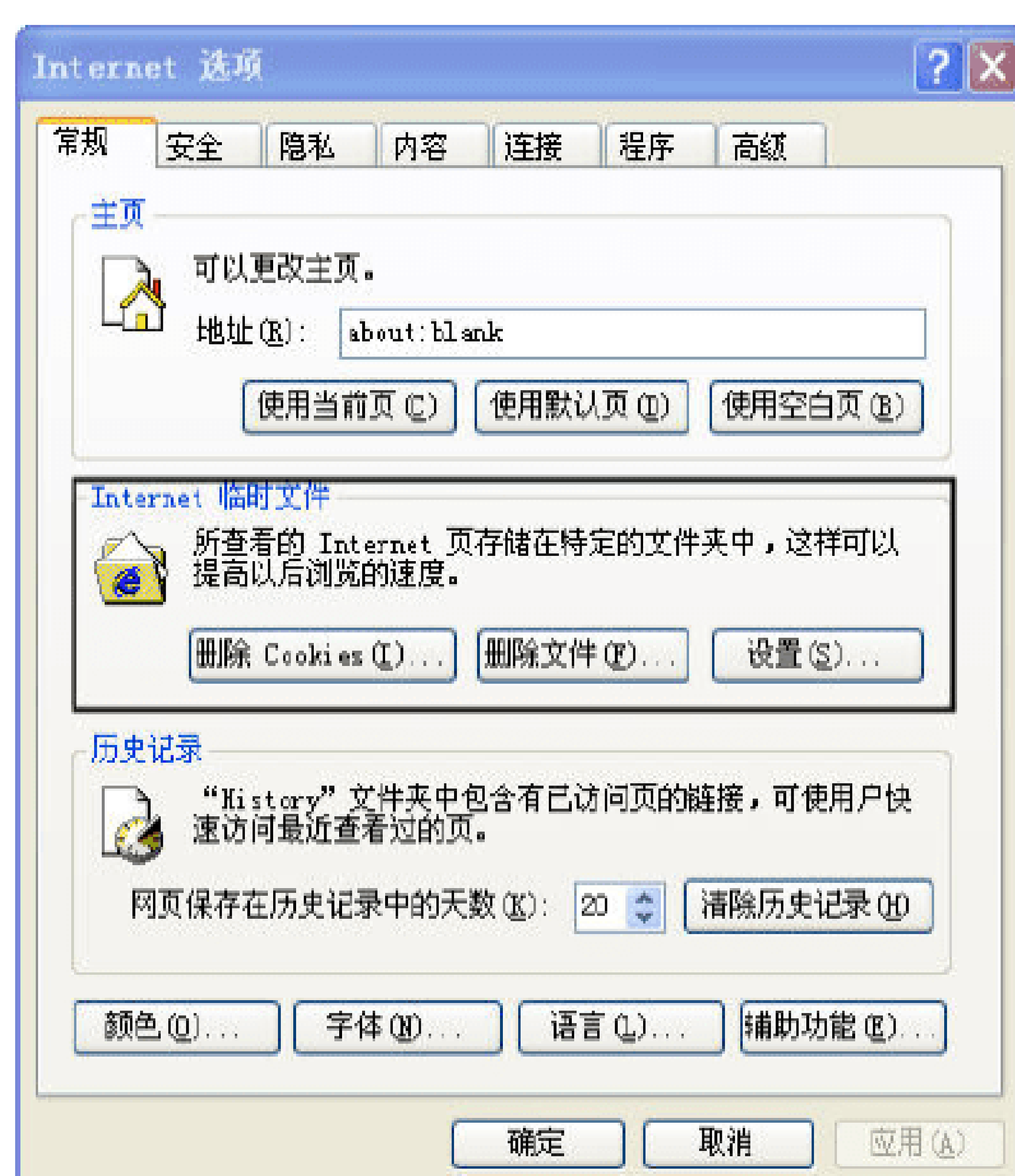


图 7.14　Internet 选项对话框

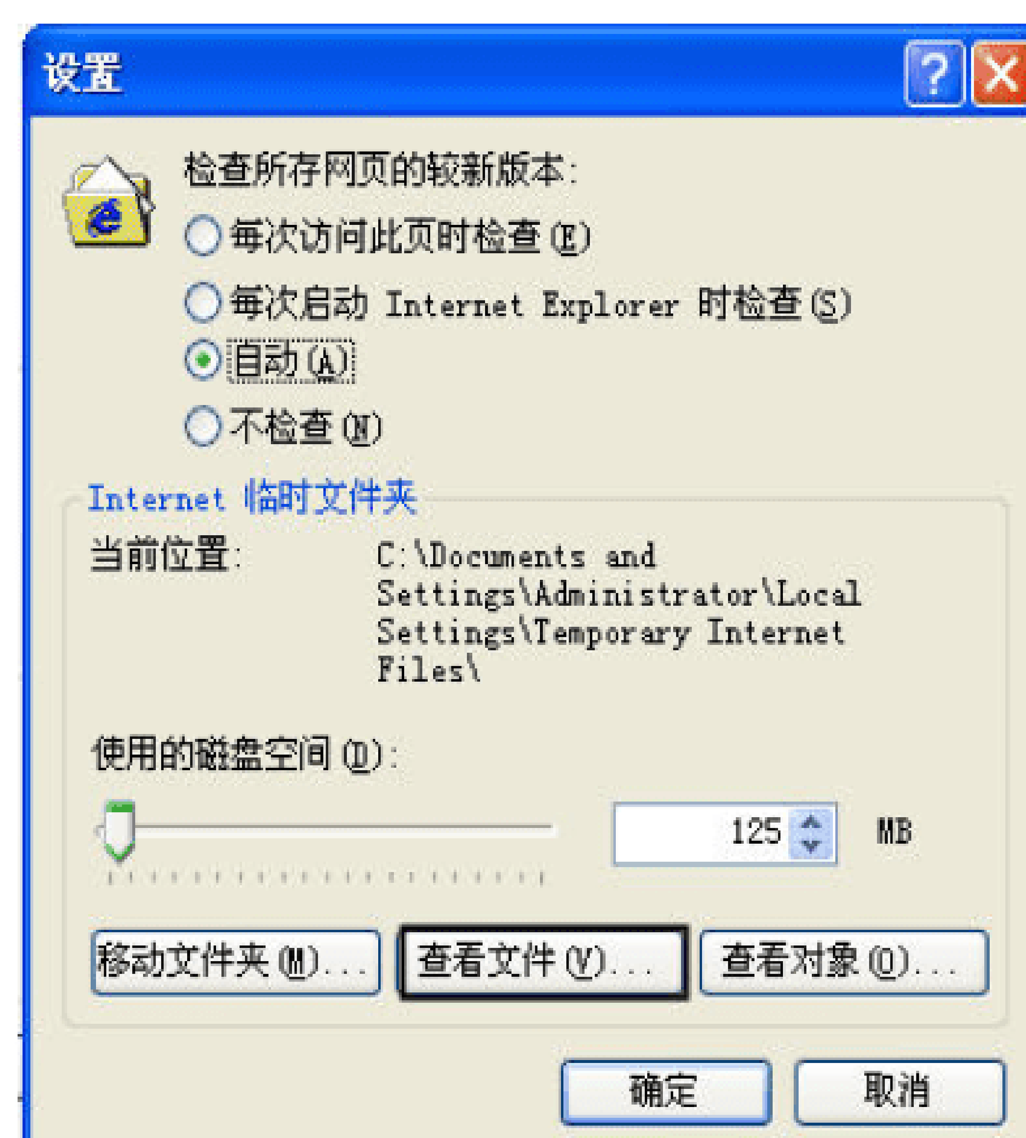


图 7.15　设置对话框

如果想让本地计算机不保存 Cookie，可以在“Internet 选项”对话框中选择“隐私”选项卡（如图 7.16 所示），把“设置”区域中的滑块移动到最上端。或者在该区域中单击“高级”按钮，在打开的“高级隐私策略设置”对话框中（如图 7.17 所示），选择拒绝第一方和第三方的 Cookie。

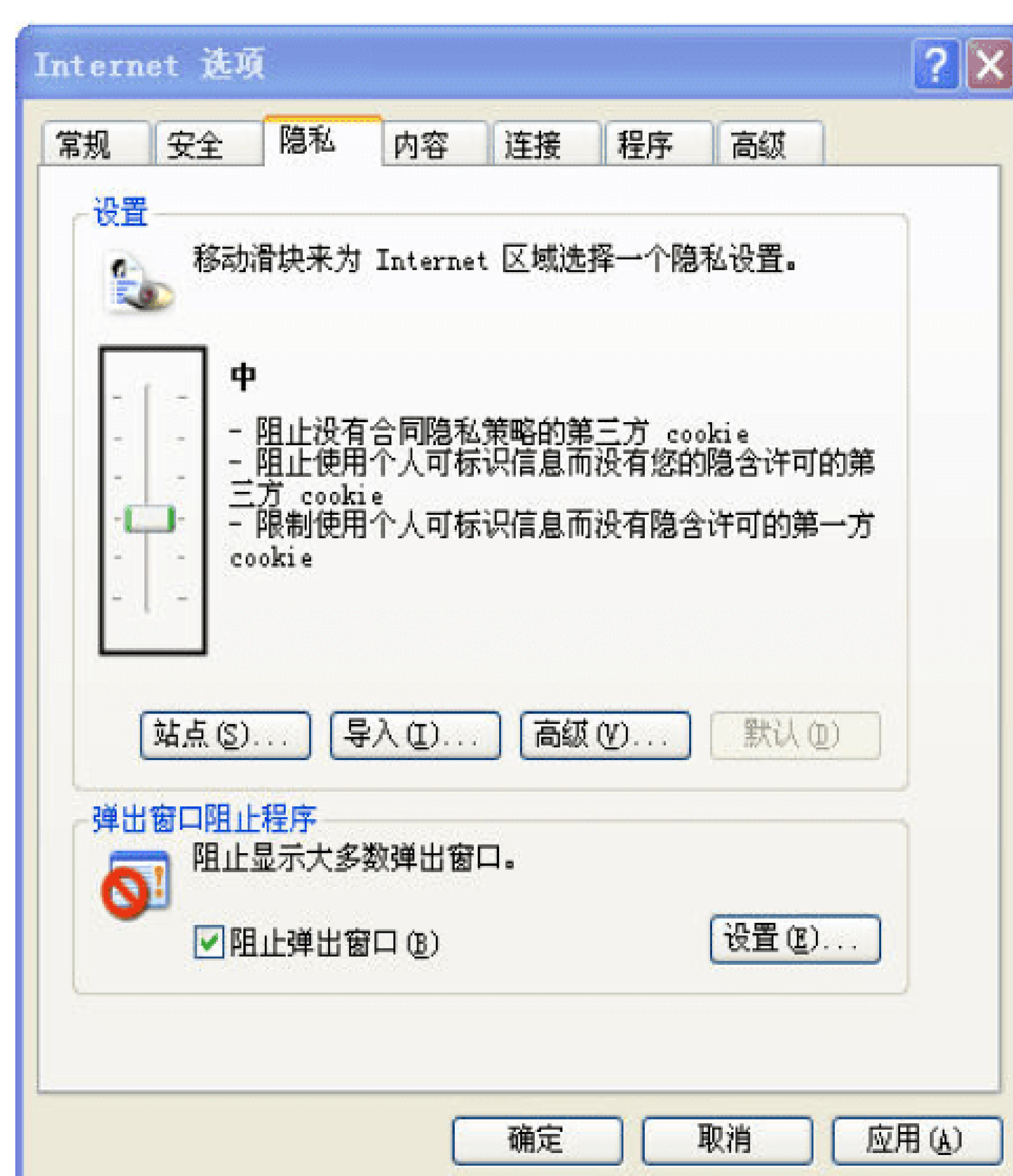


图 7.16　隐私选项卡

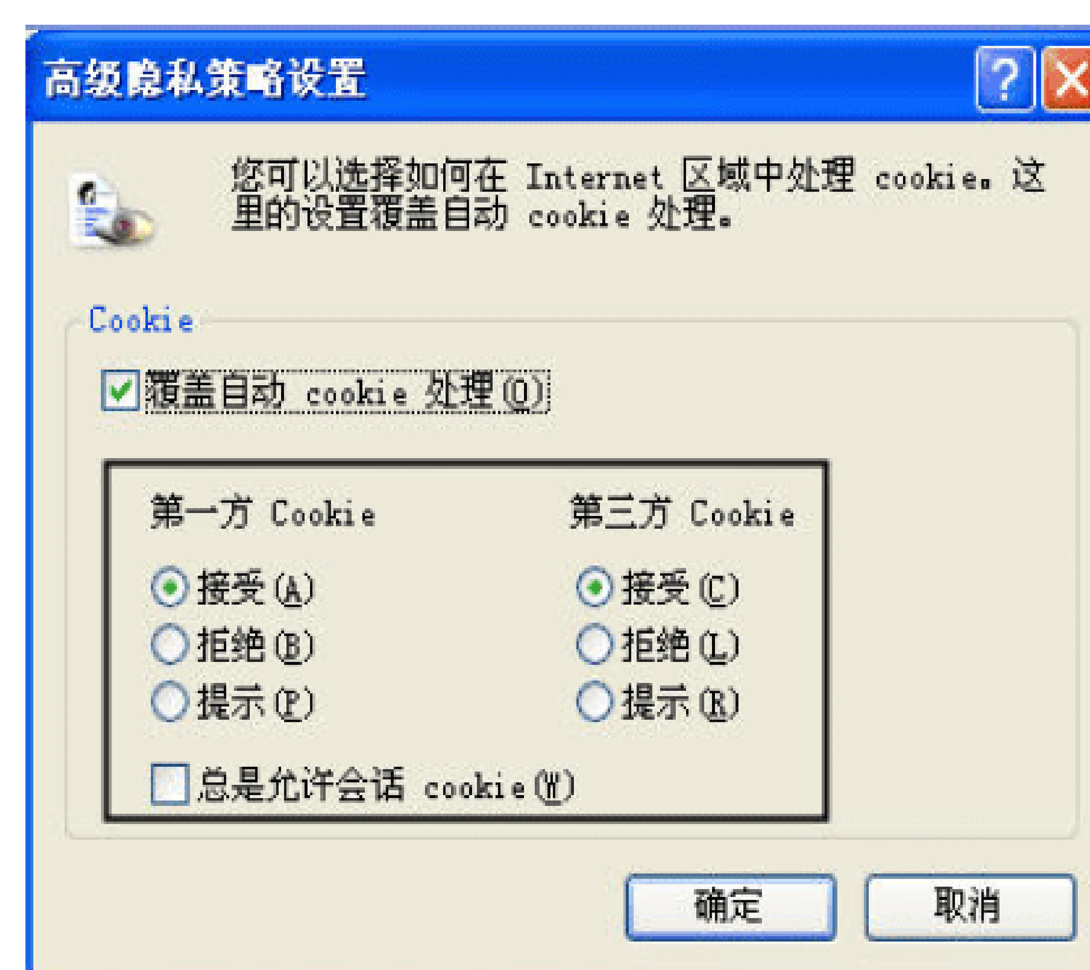


图 7.17　高级隐私策略设置

保存在本地计算机硬盘中的 Cookie，可以被所有打开的 IE 浏览器共享，但是保存在

一个 IE 浏览器进程中的 Cookie 是不能被其他打开的浏览器共享。通过选择“文件”|“新建”|窗口”命令打开的窗口可以共享原窗口的 Cookie。

注意：上述情况只是针对微软的 IE 浏览器，如果是 Mozilla Firefox 浏览器，则所有进程都共享 Cookie。

7.3.2　关于 Cookie 的响应头字段和请求头字段

所谓响应头字段用于指定 Java Web 服务器向客户端传送的 Cookie 内容，其一般会附带在响应消息头部分并且具有相应的格式。当客户端接受 Java Web 服务器发送过来的 Cookie 信息后，它将存储到客户端，并在以后对该 Java Web 服务器的每次访问请求中，都使用一个 Cookie 请求头字段将 Cookie 信息回送给 Web 服务器，这就是所谓的请求头字段。Cookie 请求头字段中的设置与从响应头字段中接受到的内容一致。

Cookie 中的内容是以键-值对的方式记录，各个键的名字和含义都要遵循规范。到目前为止，Cookie 有 3 个规范：最早的 Nescape 规范、RFC2109 规范和 RFC2975 规范，为了确保互操作性默认情况下采用 Nescape 规范来创建 Cookie。

注意：虽然在具体编程中 Nescape 规范的响应头字段比较常见，但是其与 RFC2975 规范的响应头字段的语法和作用类似，为了方便讲解，本节将对 RFC2975 规范的响应头字段进行介绍。

当 Web 服务器响应第一次访问的资源时，会在响应消息头部分附带 RFC2975 规范的响应头字段。该头字段字符串具有一定格式：

```
Set-Cookie:Name=Value;Comment=Value;Domain=Value;Max-Age=Value;Path=
Value;Secure;Version= Value
```

上述代码中各个属性的具体含义，如表 7.1 所示。

表 7.1　相关属性的值

属　性　名	意　　义
Name	设置该 Cookie 的名字，该属性必须出现在其他属性的前面同时也是必须设置的属性
Comment	该属性用来设置 Cookie 的提示信息，即浏览器会向浏览者提示设置信息，让用户自己决定是否接收该 Cookie
Domain	该属性用来设置该 Cookie 在哪个域中有效，当浏览器访问该域中的服务器时，就会被回传 Cookie
Max-Age	该属性用来设置该 Cookie 在客户端保持有效的时间，其中 Value 值是以秒为单位的十进制整数
Path	该属性用来指定该 Cookie 对服务器上的那些目录和那些 URL 目录有效。该属性的默认值是浏览器请求资源的目录
Secure	该属性用来通知浏览器回传该 Cookie 信息时，应该使用安全的方式访问服务器。该属性没有属性值。
Version	该属性用来指定该 Cookie 所遵循的版本格式，属性值 Value 为一个十进制的整数

注意：上述各属性的值是区分大小写的。

7.3.3　支持 Cookie 相关类

为了便于开发项目，Sun 公司开发出一系列的类和方法来支持 Cookie 方面的编程。查看 Servlet API 开发文档可以知道，如果想创建一个 Cookie 对象，可以使用 javax.servlet.http.Cookie 类；如果想向浏览器发送 Cookie 信息，可以使用 HttpServletResponse 接口中定义的 addCookie()方法；如果想读取浏览器回送的 Cookie 信息，可以使用 HttpServletRequest 接口中定义的 getCookies()方法。对于 Cookie 类应该掌握如下的方法和属性。

- 对于 Cookie 的构造函数：public Cookie(String name,String value)，其中的两个参数分别代表 Cookie 的名称和值。getName()方法用于返回 Cookie 的名称，setValue()和 getValue()方法分别用于设置和返回 Cookie 的值，setMaxAge()和 getMaxAge()方法分别用于设置和返回 Cookie 在浏览器计算机上存储的时间。
- 当使用 setMaxAge()方法设置时间时，如果设置的值为 0，浏览器会立即删除 Cookie；如果设置的值为负数，浏览器在关闭时才会删除 Cookie，这样 Cookie 就会保存到该浏览器的进程中；如果设置的值为正数，只有设置的时间到期才会删除 Cookie，这样 Cookie 会保存到本地计算机的硬盘中。
- 当在服务器程序中调用 HttpServletResponse 接口中的 addCookie()方法时，服务器就会在响应消息中增加一个响应头，该响应头的设置值由该方法的 Cookie 类型参数决定。在一个具体的服务器程序中可以多次调用该 addCookie()方法来设置多个响应头。
- 当在服务器程序中调用 HttpServletRequest 接口中的 getCookie()方法时，服务器就会从请求消息的请求头中读取关于 Cookie 的项，然后把这些项组成 Cookie 对象返回。

7.4　统计访问量功能

本章通过 JSP+Servlet 框架技术来实现统计访问量，统计访问量程序架构如图 7.18 所示，它包含一个 Servlet 程序和调用该程序 JSP 页面：CountFileHandler.java 和 count.jsp。

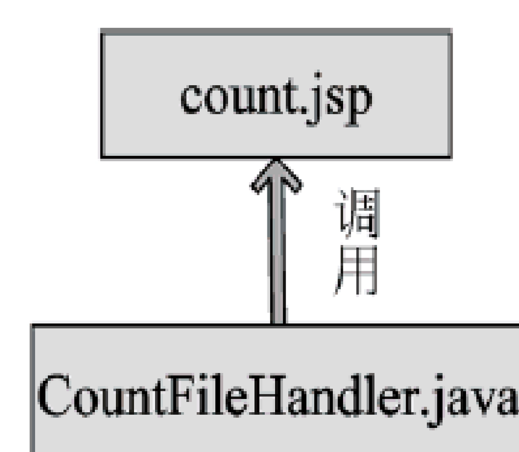


图 7.18　程序关系图

7.4.1　读取和保存访问量

CountFileHandler 这个服务器端 Servlet 程序，用来实现从文本中读取访问量并且在关闭该项目时，保存访问量到特定文本。代码 7.4 实现访问量的读取和保存。

代码 7.4　读取和保存访问量：CountFileHandler.java

```
...
```

```
public class CountFileHandler {
    //保存访问量信息到特定文本
    public static void writeFile(String filename, long count) {
        try {
            PrintWriter out = new PrintWriter(new FileWriter(filename));
                                                        //获取写入流
            out.println(count);                         //把访问量值写进文件
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //从特定的文本中读取访问量信息
    public static long readFile(String filename) {
        long count = 0;                                 //定义一个变量
        try {
            File f = new File(filename);                //创建一个 File 类型对象
            if (!f.exists()) {                          //如果文件不存在
                //则调用 writeFile()方法创建特定文件，并写入访问量的值为 0
                writeFile(filename, 0);
            }
            //创建读入流
            BufferedReader in = new BufferedReader(new FileReader(f));
            count = Long.parseLong(in.readLine()); //把读取的字符转换类型
            in.close();                                 //关闭读入流
        } catch (IOException e) {
            e.printStackTrace();
        }
        return count;
    }
}
```

【代码解析】

- CountFileHandler 程序由两个方法组成，writeFile()方法用来把新的信息（访问次数）写入特定文件，而 readFile()方法用来读取特定文件中信息（访问次数）。
- 在方法 writeFile()中，需要两个参数，filename 表示特定的文件名，而 count 表示更新后的访问次数。在具体编写时，首先获取到 filename 文件的写入流，然后把 count 写到该文件中，最后要记得关闭写入流。
- 在 readFile()方法中，只需要一个表示特定文件的 filename 参数就可以。在具体编写时，首先需要判断名为 filename 值的文件是否存在，如果该文件不存在则表示该网站第一次运行，所以应该调用 writeFile()方法创建该文件并向该文件中写入 0（访问量）。如果该文件存在，首先需要创建该文件的读取流，然后读取该文件中的内容并把其转换成数值型，最后在关闭读取流的同时返回转换后的值 count。

7.4.2　存在缺陷的获取访问量

count.jsp 这个页面程序首先利用 CountFileHandler.readFile()获取访问量信息，然后更新该信息，之后不仅显示出该信息而且还利用 CountFileHandler.writeFile()方法保存到特定文件中。代码 7.5 实现了显示访问量。

代码 7.5　显示访问量：count.jsp

```
...
<body>
<%
//把数字转换成图片
public String transform(long count){
    String countNumber=count+"";             //把 count 变量转换成字符串
    String newNumber="";
    //实现对字符串的切割
    for(int i=0;i<countNumber.length();i++){
        //转换成图片数值
        newNumber=newNumber+"<img src=\"images\\"+countNumber.charAt(i)+
        ".gif\">";
    }
    return newNumber;
}
%>
<%
    //读取 count 文件中内容，并转换成数值型
    long count=CountFileHandler.readFile(request.getRealPath("/")+
    "count.txt");
    count=count+1;                           //更新变量 count
    //把更新后的 count 值写入 count 文件中
    CountFileHandler.writeFile(request.getRealPath("/")+"count.txt",
    count);
%>
<h2>
欢迎您访问，本页面已经被访问过
<%=transform(count) %>次了。
</h2>
</body>
...
```

【代码解析】

- 在 count.jsp 这个页面程序中，不仅正确地显示出其访问量，同时利用图片代替访问量的每个具体数字来实现页面的美化。
- 在第二段 Java 代码中，主要用来实现读取旧的访问量、更新访问量和保存新访问量这三个功能。
- 在第一段 Java 代码中定义了 transform()方法，在该方法中首先把 long 型参数 count 转换成字符串以方便对其切割，然后在遍历该字符串时通过 charAt()方法获取字符串中的每个数字，同时用该数字相对应的图片组成一个新的图片数组，最后返回该图片数组。

7.4.3　改进获取访问量

count2.jsp 这个页面程序与 count.jsp 基本相同，只是比其多了限制“刷新”的功能，即当浏览者刷新该页面时，显示的访问量不会增加。代码 7.6 实现了显示访问量。

代码 7.6　显示访问量：count2.jsp

```
...
```

```
<body>
<%
public String transform(long count){
    String countNumber=count+"";
    String newNumber="";
    for(int i=0;i<countNumber.length();i++){
        newNumber=newNumber+"<img src=\"images\\"+countNumber.charAt(i)+
        ".gif\">";
    }
    return newNumber;
}
%>
<%
long
count=CountFileHandler.readFile(request.getRealPath("/")+"count.txt");
if(session.getAttribute("visited")==null){
    session.setAttribute("visited","y");
    session.setMaxInactiveInterval(70*70*24);
    count=count+1;
    CountFileHandler.writeFile(request.getRealPath("/")+"count.txt",
    count);
}
%>
<h2>
欢迎您访问，本页面已经被访问过
<%=transform(count) %>次了。
</h2>
</body>
...
```

【代码解析】

count2.jsp 这个页面程序是通过设置 Session 对象来限制“刷新”功能，即在第二段 Java 程序中在获取旧的访问量后，需要判断名为 visited 的 Session 对象。当该对象不存在时则表示该浏览者不是在刷新该网页，此时首先应该创建名为 visited 的对象，然后设置该对象的存在时间为一天。最后更新变量 count 的值，然后把该值保存到 count 文件中。否则只会使用变量 count 更新以前的值。

7.5　指点迷津——Session 知识

当使用 Cookie 技术传递的信息较多时，将会严重降低网络传输效率，增大服务器端程序处理的难度。为此在服务器端提供了一种保存会话状态的技术，即 Session 技术。所谓 Session 就是指有始有终的一系列动作/消息，比如打电话时从拿起电话拨号到挂断电话这中间的一系列过程。

7.5.1　Session 基础知识

在弥补 HTTP 无状态的缺陷时，除了可以使用 Cookie 外，还可以使用 Session。Session 的翻译为中文为“会话”，那么 Session 与 Cookie 到底有什么区别呢？

打个比方：在一些大型商场里管理员为了促进销售，对于老顾客会实行一种优惠措施：

每买够 1000 元返回 10 元。为了使措施能够成功执行，管理员可能有 3 种方案。

第 1 种方案：管理员记住每一个顾客的每一次消费，等到顾客消费满 1000 元的时候就返回 10 元。这好比 HTTP 协议本身是有状态的，可以记住顾客的活动行为。

第 2 种方案：管理员可以发给顾客一张积分卡，上面记录着消费的数量，一般还有个有效期限。每次消费时，如果顾客出示这张积分卡，就需要修改积分卡上的积分。这种做法就是在客户端保持状态，好比是 Cookie 技术。

第 3 种方案：管理员发给顾客一张会员卡，除了卡号之外什么信息也不记录，每次消费时，如果顾客出示该卡片，管理员则在本店的档案中，找到该卡的卡号进行修改。这种做法就是在服务器端保持状态，好比是 Session 技术。

从上述的第 2 种和第 3 种方案中可以发现 Cookie 和 Session 的区别。

- Cookie 是把积分卡发给顾客，信息保存在客户端；而 Session 的信息由管理员管理，保存在服务器端。由于这个区别，Cookie 的大小不能超过 4KB，而 Session 却没有这个限制。可以通过浏览器来禁止接受 Cookie，而 Session 却不能通过其禁止。
- 与 Cookie 相关联的积分卡，上面记录了关于顾客所有的消费信息；而与 Session 相关联的会员卡，上面只有一个卡号。由于这个区别，Session 比 Cookie 更安全，因为对于 Session 来说，客户端只是知道卡号（session id）。

在开发具体应用时，Session 主要用于在页面间传递变量，即该变量对用户的所有操作过程都有效。通过为每个浏览者分配一个 session id 来唯一的标识浏览者。而 Cookie 主要用于实现从客户端到服务器端或从服务器端到客户端之间变量的传递，只能是当次传递有效，而不能跨窗口。

7.5.2　支持 Session 相关类

当浏览者访问服务器端的某个 Servlet 程序时，如果该 Servlet 程序决定与浏览器开启一个会话时，服务器端就会创建一个与该浏览者相对应的 HttpSession 对象，并为该 HttpSession 对象分配一个唯一的 session id（会话标识号）。然后在响应消息中把会话标识号传递给浏览器。浏览器会记住该会话标识号，并在后续的每次访问请求中都把这个会话标识号传递给服务器，服务器会依据传递过来的会话标识号选择相对应的 HttpSession 对象。

在 HttpSession 接口中定义了各种管理和操作会话状态的方法，由其创建的 HttpSession 对象就是用来存储会话状态信息的存储结构。那么 Web 服务器如何判断 HttpSession 对象过期呢？Web 服务器无法根据 HTTP 协议判断当前浏览器是否还会继续访问该服务器，也无法检查浏览器是否关闭，所以只要关闭浏览器，Session 就消失的说法是错误的。其实 HttpSession 对象只要没有超过限定的时间段，就会一直驻留在服务器内存中，该限定的时间是在 Tomcat 安装目录\conf\web.xml 文件中设置，其代码如下：

```
<session-config>
<session-timeout>70</session-timeout >
</session-config>
```

该文件中设置的时间值是以分钟为单位，Tomcat 服务器的默认会话时间为 30 分钟。

下面介绍一些必须掌握的 HttpSession 类的方法。

- void setAttribute(String,Object)：将一对相关联的对象与名称存储进当前的HttpSession 对象中。当 HttpSession 对象中已经存在指定名称的属性时，该方法则删除原来的属性后再增加新的属性。当传递给 setAttribute()方法的属性值对象为null 时，则删除指定名称的属性。
- Object getAttribute(String)：从当前的 HttpSession 对象中返回指定名称的属性对象。在 HttpSession 接口的实现类中通常都会定义一个 HashMap 类型的成员变量，该方法就是在该成员变量中根据名称检索对象。
- void removeAttribute(String name)：删除当前 HttpSession 对象中指定名称的属性。
- Enumeration getAttributeNames()：用于返回一个包含当前 HttpSession 对象中的所有属性名的 Enumeration 对象。
- void invalidate()：强制当前 HttpSession 对象无效，即服务器可以立即释放该HttpSession 对象，而不用等到超时后才释放该 HttpSession 对象。
- int getMaxInactiveInterval()：返回当前 HttpSession 对象距离限定时间之间的时间，该时间是以秒为单位。
- void setMaxInactiveInterval(int interval)：设置当前 HttpSession 对象限制时间，即修改当前会话的默认限制时间。当某个 HttpSession 对象在超过设置的时间后还没有接收到后续的访问请求时，则该 HttpSession 对象将失效。

除了 HttpSession 接口中关于 Session 对象的函数和方法外，HttpServletRequest 接口中也定义了一些与 Session 相关的方法，之所以会出现在该方法中，因为 Session 与每个请求消息紧密相关。

在 HttpServletRequest 接口中可以通过 getSession()方法返回与当前请求相关的HttpSession 对象，其有两种重载形式：

```
public HttpSession getSession(boolean create)
public HttpSession getSession()
```

当浏览器的请求中没有 session id，第一个方法根据参数决定是否创建新的HttpSession，如果参数为 true 则创建；否则就不创建。而第二个方法则相当于第一个方法的参数为 true 的情况。

当浏览器的请求中包含有 session id，服务器检索并返回与这个 session id 对应的HttpSession 对象。

要想熟练掌握关于 Session 方面的编程，除了需要掌握 Session 编程的关键类外，还需要掌握其底层是如何实现的。

其实 Session 对象是保存在服务器的内存中，每一个浏览者都会对应相应会话对象，为了区分不同浏览者，每个浏览者都有一个编号，即 session id。根据这个 session id 就可以找到会话对象，而根据会话对象就可以找到其对应的 key 值和 value 值。如果利用 Cookie来实现底层，那么当浏览器第一次请求资源，服务器的响应头中就会写入 session id，浏览器就会把 session id 存储到本地计算机上，以后每次访问服务器就会把该值写入请求头，如图 7.19 所示。

当浏览器不接受 Cookie 时，依然可以使用 Session，就是在每次请求后面增加 jsessionid参数，例如：

```
/servlet/test.java;jsessionid=235D 9752 5120 4ECF 8DCC
```

注意：上述代码 jsessionid=235D 9752 5120 4ECF 8DCC 在具体编写时，编程人员不需要自己编写，而是通过 encodeURL()方法来实现，如下：

```
out.println(response. encodeURL("/servlet/test.java"));
```

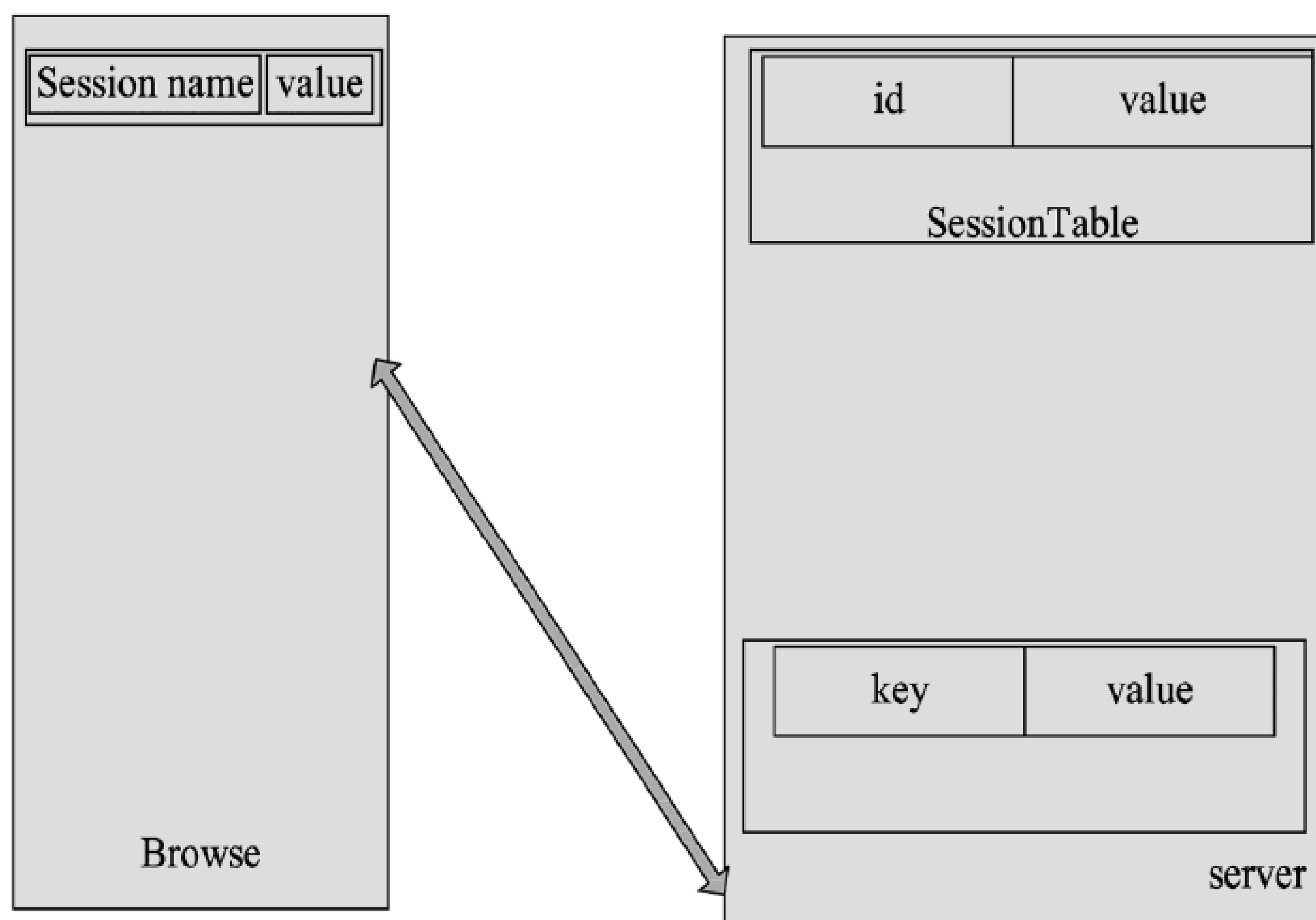


图 7.19　底层实现

7.6　统计在线人数功能

本章通过 JSP+Servlet 框架技术来实现统计访问量功能，统计访问量程序架构如图 7.20 所示，它包含一个 Servlet 程序和调用该程序 JSP 页面：CounterListener.java 和 showpeople.jsp。

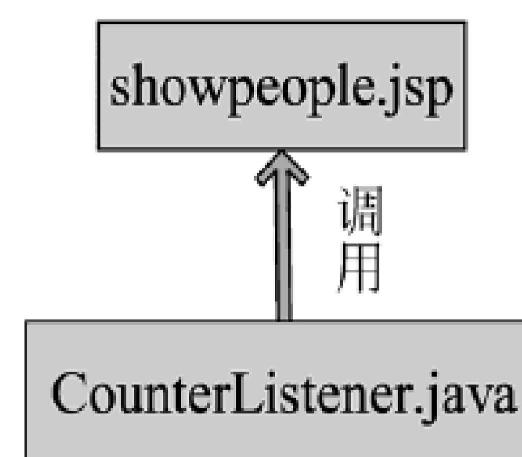


图 7.20　程序关系图

7.6.1　统计在线人数

CounterListener 服务器端 Servlet 程序，用来实现统计当前访问该网页的用户人数，即统计在线人数。代码 7.7 通过监听 Session 对象实现统计在线人数。

代码 7.7　统计在线人数：CounterListener.java

```
...
public class CounterListener implements HttpSessionListener {
    private static long onlineNumber = 0;       //定义属性
    public static long getOnlineNumber() {      //设置属性
        return onlineNumber;
    }
    public void sessionCreated(HttpSessionEvent se) {
                                                //当创建 Session 时的操作
        onlineNumber++;                         //让 onlineNumber 的值加 1
```

```
    }
    public void sessionDestroyed(HttpSessionEvent se) {
                                            //当删除 Session 时的操作
        onlineNumber--;                     //让 onlineNumber 的值减 1
    }
}
```

接着在 web.xml 文件中对该进行配置。

```
<!--配置监听器-->
<listener>
    <listener-class>
        com.cjg.CounterListener
    </listener-class>
</listener>
```

【代码解析】

CounterListener 程序要对 Session 对象实现监听，首先必须继承 HttpSessionListener 类，该程序的基本原理就是当浏览者访问页面时，必定会产生 Session 对象；当关闭该页面时，必定会删除该 Session 对象。所以每当产生一个新的 Session 对象就让在线人数就加 1，每当删除一个 Session 对象就使在线人数减 1。

showpeople.jsp 页面程序直接输出 CounterListener.getOnlineNumber()的值，就可以显示出在线人数。代码 7.8 用来显示在线人数。

代码 7.8　显示在线人数：showpeople.jsp

```
...
    <body>
        <!--显示在线人数-->
        当前应用中一共有<%=CounterListener.getOnlineNumber()%>人在线<br>
    </body>
...
```

7.6.2　多学两招——关于监听器分类

所谓监听器，就是专门用于对其他对象身上发生的事件或状态改变进行监听，一旦被监视的对象发生改变时，立即采取相应的行动。即监听器对象可以在情况发生前、发生后做一些必要的处理。

所谓 Servlet 监听器，就是 Servlet 规范中定义的一种特殊类（Listener），用于监听 Web 应用程序中的 ServletContext、HttpSession 和 ServletRequest 等域对象的创建与销毁事件，以及监听这些域对象中的属性发生修改时的事件。目前 Servlet 2.4 和 JSP 2.0 总共有 8 个监听器接口和 7 个 Event 类，如表 7.2 所示。

表 7.2　监听器接口

Listener 接口	Event 类
ServletContextListener	ServletContextEvent
ServletContextAttributeListener	ServletContextAttributeEvent
HttpSessionListener	HttpSessionEvent
HttpSessionActivationListener	HttpSessionEvent

续表

Listener 接口	Event 类
HttpSessionAttributeListener	HttpSessionBindingEvent
HttpSessionBindingListener	HttpSessionBindingEvent
ServletRequestListener	ServletRequestEvent
ServletRequestAttributeListener	ServletRequestAttributeEvent

根据监听事件的类型可以把 Servlet 监听器分成如下 3 种类型。

- 监听域对象自身的创建和销毁的事件监听器。
- 监听域对象中的属性的增加和删除的事件监听器。
- 监听绑定到 HttpSession 域中某个对象的状态的事件监听器。

使用监听器不仅方便控制 Application、Session 和 Request 对象发生的特定事件，而且还可以集中处理特定的事件。

7.6.3　多学两招——关于监听器类

在 7.6.2 节中已经介绍了监听器的定义、监听器的作用、监听器的接口类和事件类，以及监听器的分类，本节将在这些基础上接着讲如何编写监听器，具体步骤如下。

（1）Servlet 规范为每种事件监听器都定义了相应的接口，开发人员编写的事件监听器程序只需实现这些接口，web 服务器根据用户编写的事件监听器所实现的接口把它注册到相应的被监听对象上。

（2）一些 Servlet 事件监听器需要在 Web 应用程序的 web.xml 文件中进行注册，一个 web.xml 文件中可以注册多个 Servlet 事件监听器，Web 服务器按照它们在 web.xml 文件中的注册顺序来加载、注册这些 Serlvet 事件监听器。

注册一个监听程序到 web.xml 文件，需要在该文件中放置一个 listener 元素。在 listener 元素内，listener-class 元素列出监听程序的完整的限定类名，代码如下：

```
<listener>
<listener-class>package.ListenerClass</listener-class>
</listener>
```

注意： web.xml 文件中 web-app 元素内的子元素次序是 listener 元素位于所有的 servlet 元素之前以及所有 filter-mapping 元素之后。此外，serlvet 规范的版本为 2.3 而不是 2.2 版本。

（3）Serlvet 事件监听器的注册和调用过程都是由 Web 容器自动完成的，当发生被监听的对象被创建、修改或销毁时，Web 容器将调用与之相关的 Serlvet 事件监听器对象的相关方法，开发人员只需要在这些方法中编写相应的事件处理代码即可。

（4）由于一个 Web 应用程序只会为每个事件监听器创建一个对象，有可能出现多个线程同时调用同一个事件监听器对象的情况，所以在编写事件监听器类时，应考虑多线程安全的问题。

下面将对各个监听器类进行详细介绍，在具体介绍之前先要清楚域对象的创建和销毁机制。

- ServletContext 的创建是在 Web 服务器启动并加载某个 Web 应用程序时创建相应的 ServletContext 对象，销毁是在 Web 服务器关闭或卸载时为每个 Web 应用程序销毁相应的 ServletContext 对象。
- HttpSession 是在浏览器开始与服务器会话时创建，销毁是在调用 HttpSession.invalidate()、超过了 Sessionid 限制时间和服务器进程被停止的情况。
- ServletRequest 是在每次请求开始时创建，每次访问结束后销毁。

监听域对象自身的创建和销毁的事件监听器如下。

- ServletContextListener 接口用于监听 ServletContext 对象的创建和销毁事件。当 ServletContext 对象被创建时，激发 contextInitialized(ServletContextEvent sce)方法；当 ServletContext 对象被销毁时，激发 contextDestroyed(ServletContextEvent sce)方法。
- HttpSessionListener 接口用于监听 HttpSession 对象的创建和销毁。当创建一个 Session 时，激发 sessionCreated(HttpSessionEvent se)方法；当销毁一个 Session 时，激发 sessionDestroyed(HttpSessionEvent se)方法。
- ServletRequestListener 接口用于监听 ServletRequest 对象的创建和销毁。当创建一个 ServletRequest 对象时，激发 requestInitialized(ServletRequestEvent sre)方法；当销毁一个 Session 时，激发 requestDestroyed(ServletRequestEvent sre)方法。

所谓域对象中属性变更的事件监听器，就是用来监听 ServletContext、HttpSession 和 HttpServletRequest 这 3 个对象中的属性变更信息事件的监听器，它们分别为 ServletContextAttributeListener、HttpSessionAttributeListener 和 ServletRequestAttributeListener，这 3 个接口中都定义了 3 个方法来处理被监听对象中的属性的增加、删除和替换的事件，同一个事件在这 3 个接口中对应的方法名称完全相同，只是接受的参数类型不同。当增加一个属性时，web 容器就调用事件监听器的 attributeAdded()方法进行响应。各个域属性监听器中的完整语法定义为：

```
public void attributeAdded(ServletContextAttributeEvent scae)
public void attributeReplaced(HttpSessionBindingEvent  hsbe)
public void attributeRmoved(ServletRequestAttributeEvent srae)
```

当删除一个属性时，Web 容器就调用事件监听器的 attributeRemoved()方法进行响应。各个域属性监听器中的完整语法定义为：

```
public void attributeRemoved(ServletContextAttributeEvent scae)
public void attributeRemoved (HttpSessionBindingEvent  hsbe)
public void attributeRemoved (ServletRequestAttributeEvent srae)
```

当替换一个属性时，Web 容器就调用事件监听器的 attributeReplaced()方法进行响应。各个域属性监听器中的完整语法定义为：

```
public void attributeReplaced(ServletContextAttributeEvent scae)
public void attributeReplaced (HttpSessionBindingEvent  hsbe)
public void attributeReplaced (ServletRequestAttributeEvent srae)
```

监听绑定到 HttpSession 域中的某个对象的状态的事件监听器如下：

所谓监听绑定到 HttpSession 域中的某个对象，就是实现监听器 HttpSession-BindingListener 的对象。当对象绑定到 Session Attribute 时，Web 容器就调用事件监听器的

valueBound(HttpSessionBindingEvent event)方法进行响应。当对象从 Session Attribute 对象解除绑定时，Web 容器就调用事件监听器的 valueUnbound(HttpSessionBindingEvent event)方法进行响应。

最后还有一个名为 HttpSessionActivationListener 的监听器接口，其主要用于同一个 Session 对象转移至不同的 JVM 的情形。当 Session 对象为了资源利用或负载平衡等原因而必须暂时储存至硬盘或其他储存器时，就会激发 SessionDidActivate(HttpSessionEvent se)方法；当硬盘或储存器上的 Session 对象重新加载 JVM 时，就会激发 SessionWillPassivate(HttpSessionEvent se)方法。

7.7　小　　结

本章主要介绍了统计模块，统计访问量、统计在线人数和显示欢迎信息。在具体实现显示欢迎信息模块中，使用了 Servlet 技术中的 Cookie 对象。在实现统计访问量模块中，使用了 Servlet 技术中的 Session 对象。最后利用 Servlet 技术中的监听器来实现统计在线人数功能。

在本章中不仅实现了各种模块，而且还详细介绍了 Servlet 技术中的 Cookie 对象、Session 对象和监听器。

第 8 章　网络购物车（JSP+Servlet+JavaBean）

网络购物车模块对于网上购物系统、网络购物车系统和网上开店等系统来说是一个非常重要和常见的模块，其最重要的功能就是当购买者浏览完商品后，在结账的时候显示出购买者所选的商品。

本章将通过 JSP+Servlet+JavaBean 框架技术来介绍如何实现网络购物车模块，该模块主要包含 3 种功能：浏览商品、购物和结账。

8.1　网络购物车原理

网络购物车的功能与现实中超市里购物车的功能一样，不同的只是一个是实体车而另一个是虚拟车。当购买者在购物网站上购买产品时，只需单击购买商品相关按钮就会自动保存到网络购物车里。

8.1.1　网络购物车结构框架分析

对于一个大型网上购物系统来说，实现一个可用的网络购物车功能要考虑的情况十分复杂，例如如何让购买者在浏览商品时尽可能多的购买商品、在购买商品和结账时尽可能的方便和人性化。该章将会实现一个比较简单可用的网络购物车，读者可以根据自己的需求自行进行完善。本系统的结构框架如图 8.1 所示，其项目目录如图 8.2 所示。

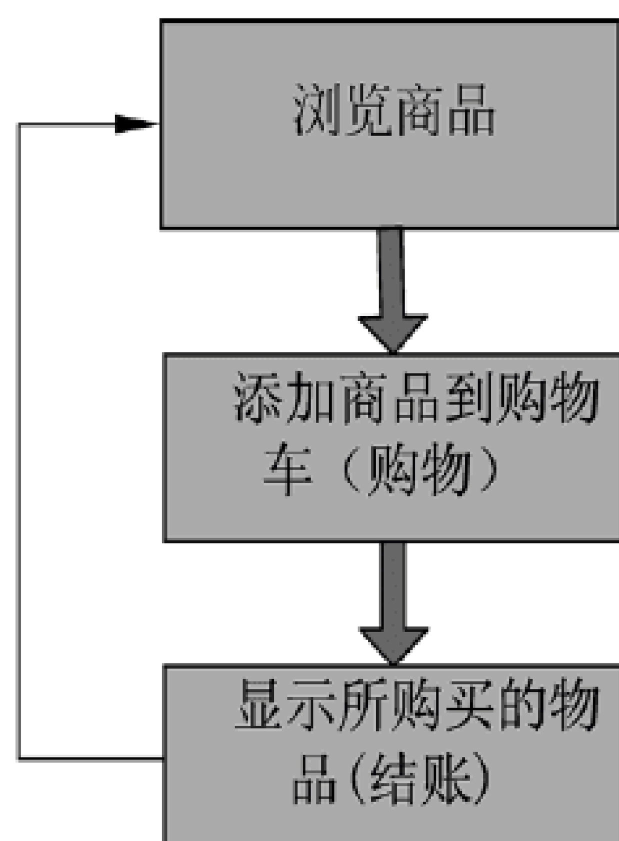


图 8.1　系统流程图

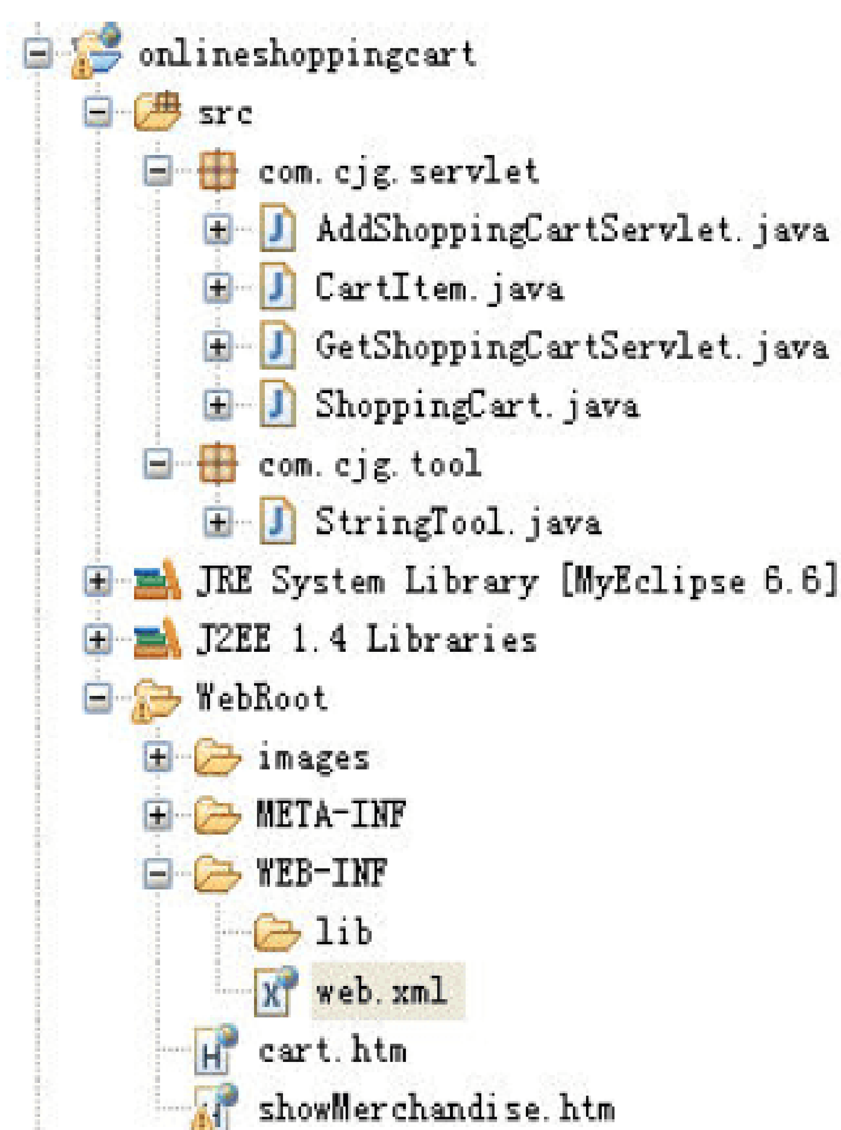


图 8.2　项目目录

8.1.2　网络购物车功能业务分析

在本节中，以直观的方式来向读者介绍整个网络购物车模块要实现的功能。这些功能包括浏览商品、购买商品和结账。

1．浏览商品

购买者首先通过浏览网页来选择所需要的商品。在 displayItems.jsp 页上有几种商品品种可以供用户选择，如 HP 笔记本电脑、移动硬盘、鼠标、LCD 显示器、电子词典和掌上游戏机，如图 8.3 所示。

2．购买商品

当购买者浏览商品时，看中了一种商品，就可以把这种商品存放到自己购物车里。每一个商品都有一个“购买”按钮提供将商品存放购物车的功能。单击移动硬盘图片下面的“购买”按钮，可把该商品放入购物车，如图 8.4 所示。



图 8.3　浏览商品



图 8.4　购买商品

3．购买商品成功

当购买者成功地把商品存放到自己购物车里，这时就会转到显示购物信息的页面，如图 8.5 所示。如果还想接着购买其他商品，可以单击“继续浏览商品，添加商品到购物车”链接转到浏览商品页面，如图 8.6 所示。

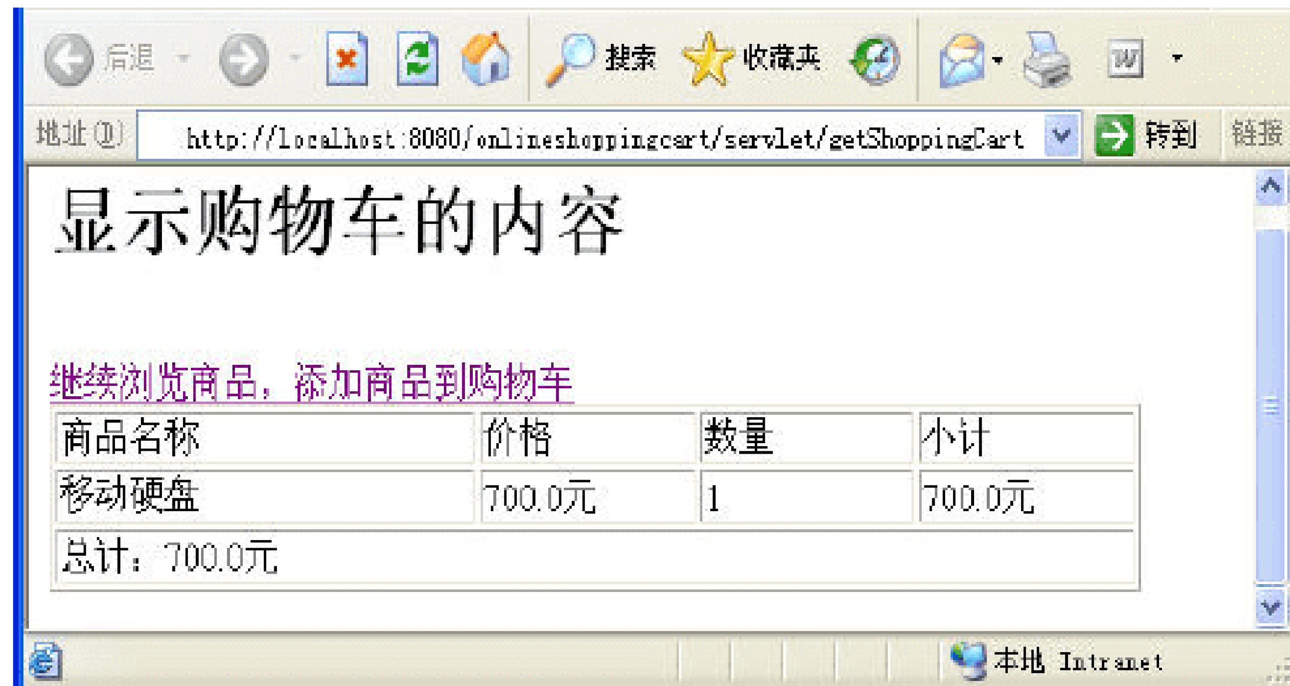


图 8.5　购买商品成功



图 8.6　继续购买商品

4．购买商品失败

如果购买者没有成功地把商品存放到自己购物车里，这时就会转到显示错误信息的页面，如图 8.7 所示。如果还想接着购买其他商品，这时可以单击该页面的“继续浏览商品，添加商品到购物车”链接如图 8.8 所示，转到浏览商品页面。



图 8.7　购买商品失败

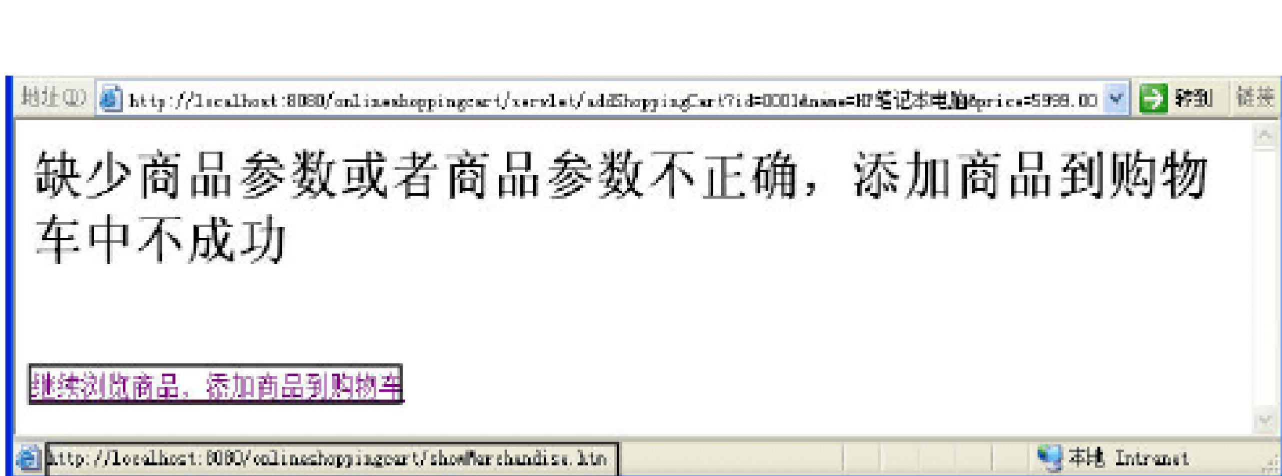


图 8.8　继续购买商品

8.2　实现网络购物车功能

本章通过 JSP+Servlet+JavaBean 框架技术来实现网络购物车的功能。网络购物车程序架构如图 8.9 所示，它包含一个 JSP 页面、两个 JavaBean 程序和两个 Servlet 程序：showMerchandise.jsp、ShoppingCart.java、CartItem.java、AddShoppingCartServlet.java 和 GetShoppingCartServlet.java。

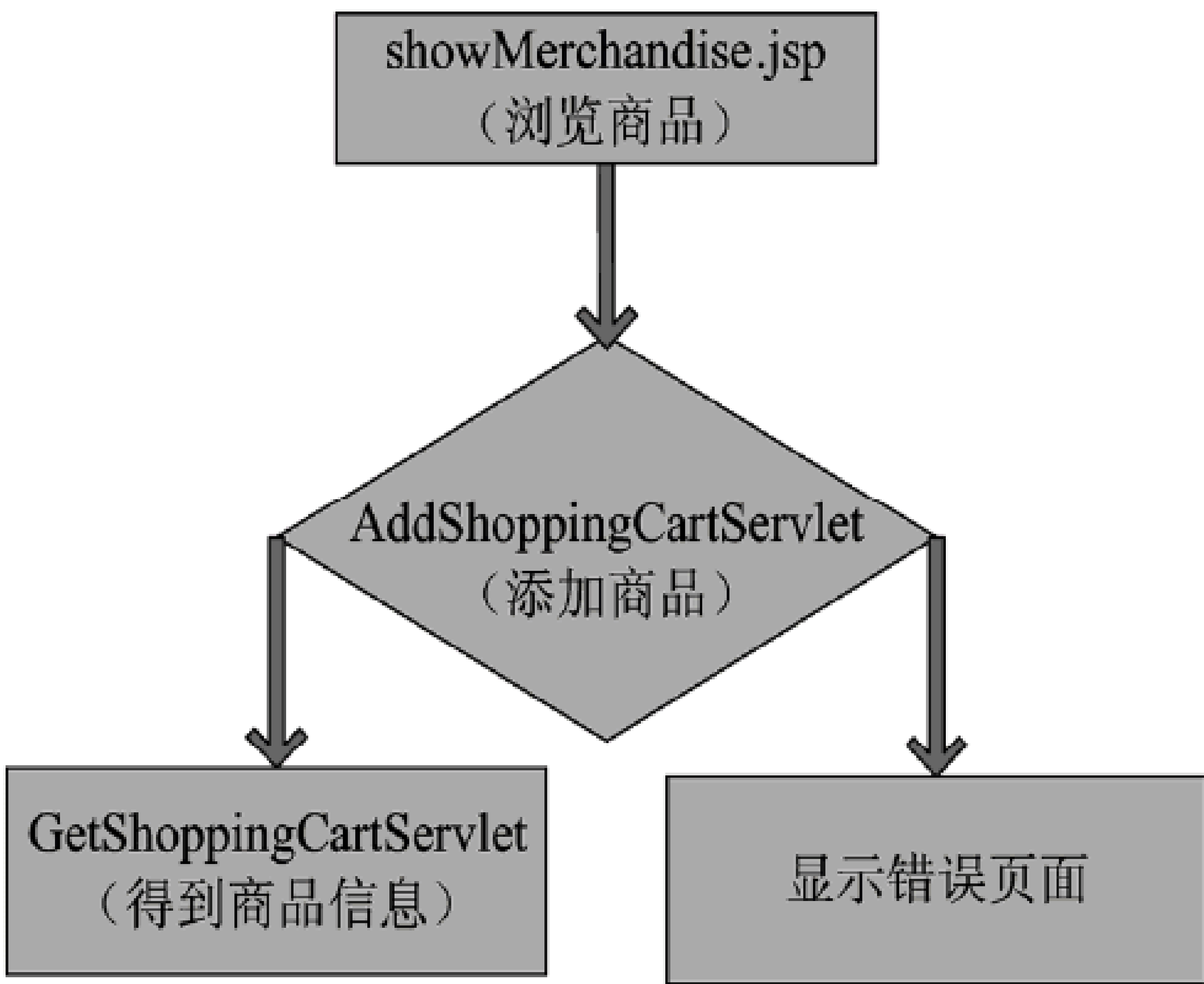


图 8.9　程序关系图

8.2.1　浏览商品网页

showMerchandise.jsp 显示商品的页面，用来让购买者浏览商品。当购买者看中某个商品决定购买后，商品就会自动存放到购物车里然后转到显示购物车里商品信息的网页，反之如果存放不成功，就会转到失败页面。代码 8.1 演示了如何实现浏览商品的功能。

代码 8.1　浏览商品：showMerchandise.jsp

```
...
```

```
<body>
    <center>
        商品显示页，请选购您喜欢的商品
    </center>
    <table width="424" height="583" border="0" align="center">
        <tr>
            <td width="190">
                <img src="images/hppavilion.jpg" width="150" height=
                "150" />
            </td>
            <td width="224">
                <img src="images/lenovo.jpg" width="150" height="150" />
            </td>
            <td>
                HP 笔记本电脑
            </td>
            <td>
                移动硬盘
            </td>
            <td>
                价格：5999.00 元
            </td>
            <td>
                价格：700.00 元
            </td>
            <!--传递“HP 笔记本电脑”的参数-->
            <<a href="/onlineshoppingcart/servlet/addShoppingCart?
            id=0001&name=HP 笔记本电脑&price=5999.00">
            <!--传递“移动硬盘”的参数-->
            <a href="/onlineshoppingcart/servlet/addShoppingCart?
            id=0002&name=移动硬盘&price=700.00">
...
    </table>
</body>
...
```

【代码解析】

- ❑ 首先页面中显示的商品价格实际上应该动态地从数据库中获取，为了方便编写，这些都编写成静态性质。
- ❑ 链接参数分为两个部分，“？”前的内容为所要转移到地址，而“？”后面则为传递的参数，这些参数必须用“&”连接。

🔔注意：*在实际编写网络购物车时，这些传递的参数值都应该是动态地从数据库中获取。*

8.2.2　商品和购物车

showMerchandise.jsp 页面上展示的每个商品信息，最好是绑定到一个 JavaBean 对象上，方便使用，CartItem.java 就是封装商品信息的 JavaBean。代码 8.2 用来设计商品类。

代码 8.2　商品类：CartItem.java

```
...
```

```
public class CartItem {
    private String name;                        //定义了商品的名称
    private int quantity;                       //定义了商品的数目
    private double price;                       //定义了商品的价格
    private String id;                          //定义了商品的类型
    private String desc;                        //定义了商品的描述
    //定义构造函数
    public CartItem(String id, String name, int quantity, double price) {
        this.id = id;
        this.name = name;
        this.quantity = quantity;
        this.price = price;
    }
    //省略 name、quantity、price、id 和 desc 属性的 get()和 set()方法
...
}
```

【代码解析】

- CartItem.java 这个 JavaBean 实际上代表的是一种商品而不是一个商品，这样设计有一个好处就是，当把多个同类商品放到购物车时，只要修改该类商品对象的个数属性即可，而不需要建立相应数目的商品对象。
- 在 CartItem.java 类中设计了 5 个属性：id 代表商品的类型、quantity 代表商品的数目、price 代表商品的价格、name 代表商品的名字，以及 desc 代表商品的描述。

对于网络购物车这个虚拟对象最好也能对应到一个 JavaBean 对象，在本项目中为 ShoppingCart.java。代码 8.3 设计网络购物车类。

代码 8.3　网络购物车类：ShoppingCart.java

```
...
public class ShoppingCart {

    private ArrayList<CartItem> cart;       //保存所有 CartItem 对象的容器对象
    public ShoppingCart() {         //定义了构造函数
        cart = new ArrayList<CartItem>();
    }
    public ArrayList<CartItem> getCart() {
                                    //返回包括所有已经购买的商品信息的容器对象
        return cart;
    }
    public void addCartItem(CartItem item) {    //添加商品到购物车里
        CartItem oldItem = null;                //定义了一个商品变量
        if (item != null) {                     //当购物车里该商品不为空
            //遍历购物车里的商品
            for (int i = 0; i < cart.size(); i++) {
                oldItem = cart.get(i);          //取出商品
                //当所要添加的商品类型与购物车里的商品类型相同
                if (oldItem.getId().equals(item.getId())) {
                    //修改商品的数量
                    oldItem.setQuantity(oldItem.getQuantity() + item.
                    getQuantity());
                    return;
                }
            }
        //当购物车里该商品为空
```

```
            cart.add(item);                         //添加商品到购物车里
        }
    }
    public boolean removeCartItem(String id) {      //从购物车中，删除商品
        CartItem item = null;                       //定义了一个商品变量
        for (int i = 0; i < cart.size(); i++) {     //遍历购物车里的商品
            item = cart.get(i);                     //取出购物车里的商品
            //当所要删除的商品类型与购物车里商品的类型相同时
            if (item.getId().equals(id)) {
                cart.remove(i);                     //删除该类型的商品
                return true;
            }
        }
        return false;
    }
    public double getTotal() {                      //计算所购所有商品的总价
        Iterator<CartItem> it = cart.iterator();    //获取商品集合
        double sum = 0.0;                           //定义了一个价格变量
        CartItem item = null;                       //定义了一个商品变量
        while (it.hasNext()) {                      //遍历商品
            item = it.next();
            sum = sum + item.getSum();
        }
        return sum;
    }
}
```

【代码解析】

- ❑ 在 ShoppingCart.java 代码中主要完成网络购物车里商品的添加（addCartItem()）、删除（removeCartItem()）和购物车信息的获取（getCart()）。
- ❑ 购物车最基本的功能就是能够装载各种商品，用代码来实现就是能够存储对象，这就需要定义一个存储 CartItem 类型对象的 ArrayList 类型的数据结构，来作为容器。即

```
Private ArrayList<CartItem> cart;
```

- ❑ 当添加一个商品到购物车中时，如果这种商品在购物车中已经存在，则修改该商品的数量，否则就把这个新的商品对象添加到容器中。
- ❑ 从购物车里删除一个商品是根据商品的 Id 号来实现的，当删除成功时，返回 true；否则返回 false。
- ❑ 在 ShoppingCart.java 代码中还实现了一个计算购买所有商品总价的函数 getTotal()，其是通过获取每种商品的数量和单价然后两者相乘来实现，该功能在获取购物车信息时使用。

8.2.3　添加商品到购物车

当购买者看中某个商品决定购买后，只要单击一下相应按钮就会存放到购物车里，该功能主要是通过服务器端程序 AddShoppingCartServlet 来实现，该程序的流程如图 8.10 所示。代码 8.4 演示了如何实现添加商品到购物车。

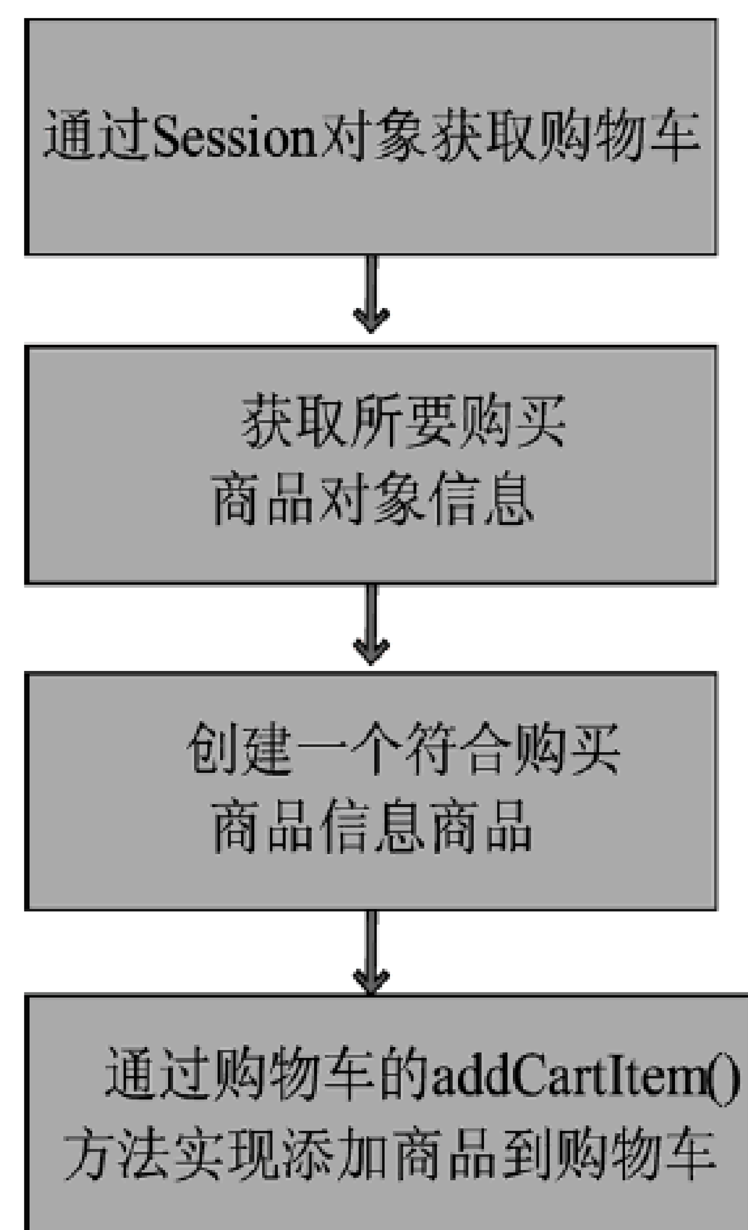


图 8.10　添加商品流程

代码 8.4　添加商品到购物车：AddShoppingCartServlet.java

```
...
public class AddShoppingCartServlet extends HttpServlet {
    private static final long serialVersionUID = -6552354364752194751L;
    //编写 doGet()方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        doPost(request, response);                     //调用 doPost()方法
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");         //设置编码方式
        HttpSession session = request.getSession();    //获取 Session 对象
        //得到购物车
        ShoppingCart cart = (ShoppingCart) session.getAttribute
        ("shoppingcart");
        //当购物车为空时
        if (cart == null) {
            cart = new ShoppingCart();                 //新建一个购物车
            session.setAttribute("shoppingcart", cart); //设置 Session 对象
        }
        String id = request.getParameter("id");       //获取所要添加商品的类型
        String name = request.getParameter("name");  //获取所要添加商品的名称
        String quantity = request.getParameter("q");
                                                    //获取所要添加商品的数目
        String price = request.getParameter("price");
                                                    //获取所要添加商品的价格
        //检验所要加入的商品信息
        if (StringTool.validateNull(id) || StringTool.validateNull(name)
        || StringUtil.validateNull(price)) {
            printError(request, response);
            return;
        }
        id = StringTool.filterHtml(id);
        name = StringUtil.filterHtml(name);
```

```
        try {
            if (StringTool.validateNull(quantity)) {
                //根据所传入商品信息创建商品，然后利用 addCartItem()方法添入购物车
                cart.addCartItem(new CartItem(id, name, 1, Double.
                parseDouble(price)));
            } else {
                //根据所传入商品信息创建商品，然后利用 addCartItem()方法添入购物车
                cart.addCartItem(new CartItem(id, name, Integer.parseInt
                (quantity), Double.parseDouble(price)));
            }
        } catch (NumberFormatException e) {
            printError(request, response);
            return;
        }
        response.sendRedirect("/onlineshoppingcart/servlet/getShopping-
        Cart");
    }

    //当添加商品失败时，会转到的地址
    private void printError(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
            IOException {
        response.setContentType("text/html;charset=UTF-8");
                                                    //设置页面的编码方式
        PrintWriter out = response.getWriter();    //得到输出流
        //错误页面
        out.println("<html>");
        out.println("<head><title>add items error</title></head>");
        out.println("<body>");
        out.println("<h1>缺少商品参数或者商品参数不正确，添加商品到购物车中不成功
        </h1><br><br>");
        out.println("<a href=\"/onlineshoppingcart/showMerchandise.
        htm\">继续浏览商品，添加商品到购物车</a><br>");
        out.println("</body>");
        out.println("</html>");
        out.flush();                                //关闭缓冲
        out.close();                                //关闭输出流
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 AddShoppingCartServlet 类路径-->
<servlet>
  <servlet-name>AddShoppingCartServlet</servlet-name>
  <servlet-class>com.cjg.servlet.AddShoppingCartServlet</servlet-class>
</servlet>
<!--配置 AddShoppingCartServlet 映射路径 -->
 <servlet-mapping>
  <servlet-name>AddShoppingCartServlet</servlet-name>
  <url-pattern>/servlet/addShoppingCart</url-pattern>
</servlet-mapping>
```

【代码解析】

- 在通过 Session 对象获取购物车时，如果获取的购物车为空则表示购买者是第一次购买商品，需要创建一个新的购物车，否则使用获取到的购物车。

- ❑ 在添加商品到购物车里时，首先要获取所要购买的商品信息，然后根据该信息创建一个商品对象，最后通过购物车对象的 addCartItem()方法实现添加功能。
- ❑ 在 AddShoppingCartServlet.java 代码中，当添加商品成功后会通过 sendRedirect()方法转移到显示购物车信息的页面，否则就会转到失败页面。

8.2.4　显示购物车商品信息

当购买者浏览完网站选定商品后，就需要结账。在结账的时候，购买者需要知道自己一共需要付多少钱和购买了一些什么商品。该功能就需要显示购物车商品情况的服务器程序 GetShoppingCartServlet.java 来实现，其流程如图 8.11 所示。代码 8.5 演示了如何实现显示购物车商品信息。

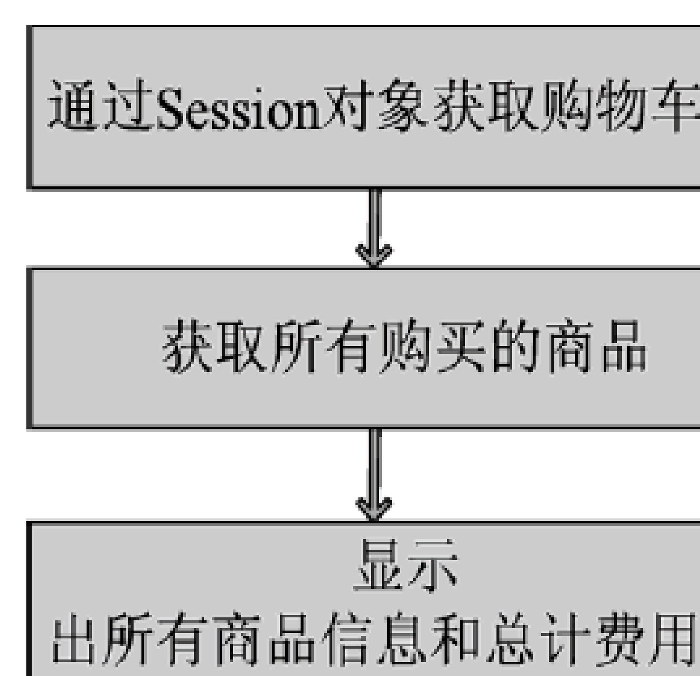


图 8.11　显示商品购物车信息流程

代码 8.5　显示商品购物车信息：GetShoppingCartServlet.java

```
...
public class GetShoppingCartServlet extends HttpServlet {
    private static final long serialVersionUID = 7724740544690435967L;
    //编写 doGet()方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        doPost(request, response);                    //调用 doPost()方法
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        HttpSession session = request.getSession();    //获取 Session 对象
        //获取购物车
        ShoppingCart cart = (ShoppingCart) session.getAttribute
        ("shoppingcart");
        response.setContentType("text/html;charset=UTF-8"); //设置页面编码
        PrintWriter out = response.getWriter();    //获取输出流
        out.println("<html>");                     //输出的页面
        out.println("<head><title>display shoppingcart</title></head>");
        out.println("<body>");
        if (cart == null) {                        //当购物车为空时
            out.println("<h1>您的购物车为空</h1><br><br>");
            out.println("<a href=\"/onlineshoppingcart/showMerchandise.
            htm\">浏览商品，添加商品到购物车</a><br>");
            return;
        }
        else {                                     //当购物车不为空时
            out.println("<h1>显示购物车的内容</h1><br>");
            out.println("<a href=\"/onlineshoppingcart/showMerchandise.
            htm\">继续浏览商品，添加商品到购物车</a>");
            printCartItem(out, cart);              //显示出购物车商品的信息
        }
        out.println("</body>");
```

```
        out.println("</html>");
        out.flush();
        out.close();
    }
    //显示购物车商品信息方法
    private void printCartItem(PrintWriter out, ShoppingCart cart) {
                                            //定义显示出购物车商品的信息的函数
        ArrayList<CartItem> items = cart.getCart(); //获取购物车里的商品容器
        CartItem item = null;               //定义一个商品对象
        out.println("<table width=\"500\" border=\"1\" align=\"left\">");
                                            //定义表格
        out.println("<tr>");
        out.println("<td width=\"200\">商品名称</td>");
        out.println("<td width=\"100\">价格</td>");
        out.println("<td width=\"100\">数量</td>");
        out.println("<td width=\"100\">小计</td>");
        out.println("</tr>");
        for (int i = 0; i < items.size(); i++) {   //遍历容器
            item = items.get(i);                  //得到商品
            out.println("<tr>");
            out.println("<td>" + item.getName() + "</td>");//获取商品的名称
            out.println("<td>" + item.getPrice() + "元</td>");
                                                  //获取商品的价格
            out.println("<td>" + item.getQuantity() + "</td>");
                                                  //获取商品的数目
            out.println("<td>" + item.getSum() + "元</td>");
                                                  //获取商品的价格
            out.println("</tr>");
        }
        out.println("<tr>");
        //输出购物车里所有商品的总价格
        out.println("<td colspan=\"4\">总计: " + cart.getTotal() + "元
        </td>");
        out.println("</tr>");
        out.println("</table>");
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 GetShoppingCartServlet 类路径-->
<servlet>
  <servlet-name>GetShoppingCartServlet</servlet-name>
  <servlet-class>com.cjg.servlet.GetShoppingCartServlet</servlet-class>
</servlet>
<!--配置 GetShoppingCartServlet 映射路径 -->
<servlet-mapping>
  <servlet-name>GetShoppingCartServlet</servlet-name>
  <url-pattern>/servlet/getShoppingCart</url-pattern>
</servlet-mapping>
```

【代码解析】

- 在通过 Session 对象获取购物车时，如果获取的购物车为空则转向出错页面，否则使用获取到的购物车。
- 在 GetShoppingCartServlet.java 代码中，编写了一个函数 printCartItem()来显示购物车商品的信息。在该函数中首先通过购物车对象 cart 获取购物车容器 items，购买

者选择的所有商品对象都存储在该容器中。最后遍历容器对象中的各个对象，并打印出每个对象的信息。

8.3　小　　结

本章主要介绍了网络购物车模块，该模块基于 JSP+Servlet+JavaBean 解决方案，保持了良好的 Java EE 中 MVC 分层思想。网络购物车在业务上主要分成两部分：添加商品到购物车和显示购物车商品。在具体实现各部分功能之前，创建了商品和购物车的相关 JavaBean 对象。

第 9 章　搜索引擎（Lucene+Web Spider）

在资源非常丰富的网络环境中，如何有效地搜索信息是一件困难的事情。为了解决该问题，于是出现了搜索引擎，例如 Google 和 Baidu 等搜索引擎。搜索引擎是一个比较特殊的技术行业，即技术门槛非常高。在许多商业项目中都需要搜索功能，一般比较常用的是使用 SQL 语言实现搜索。但是在实现全文搜索时，除了选择购买 Google 等公司的搜索服务外，还可以使用 Lucene 全文搜索组件来开发。

本章将详细地介绍如何利用 Lucene 和 Web Spider 两个组件实现全文搜索的各个方面：关于 Lucene 和 Web Spider 两个组件的简介；如何为文件建立索引；如何实现查询和如何搜索网络信息等。

9.1　关于搜索引擎的基本概念

搜索引擎英文为 Search Engine，其首先会根据一定的策略和方法，运用特定的软件程序搜集互联网上的信息，然后对信息进行组织和处理，最后将处理后的信息显示给浏览者。所谓搜索引擎系统是指为浏览者提供检索服务的系统。

9.1.1　关于搜索引擎描述

搜索引擎没有一个明确和精确的定义，一般以其发展中的一些里程碑式应用为标记划分为 3 个阶段。

- 第 1 阶段搜索引擎：该搜索引擎以“雅虎”为代表，主要依靠于人工分拣的分类目录进行搜索。
- 第 2 阶段搜索引擎：该搜索引擎以 Google 为代表，主要依靠于机器抓取和采用链接分析技术进行搜索。与第一阶段的搜索引擎相比，其信息量大、更新及时，返回信息丰富。
- 第 3 阶段搜索引擎：该搜索引擎以“综合信息搜索服务”为代表，主要在第 2 阶段的基础上加入了智能化、人机交互、自动分类技术、中文内容分析等技术，不仅提高了信息检索速度和更新频率，而且还实现了拼音纠错、模糊查询、语音查询等功能。

搜索引擎按照实现的方式，大致可以分成两大类：全文搜索引擎和分类目录搜索引擎。所谓全文搜索引擎一般通过网络机器人或网络蜘蛛工具，自动分析网络上的各种链接并将分析结果按规则整理，并同时存入数据库供显示使用。所谓分类目录搜索引擎则是通过人工的方式收集整理网站资料形成数据库。

在计算机上要表示信息获取流程，具体包含：信息的表示、信息存储、信息组织和信息访问。

（1）首先需要创建进行检索的数据，用其构造文本数据库。

（2）创建好文本数据库后，就需要建立文档的索引。Lucene 全文搜索组件中是通过倒排索引的方法来创建索引。

（3）创建好索引后，就可以进行检索。用户首先需要给出一个查询，该查询将被分析、然后利用文本处理技术进行处理。

（4）最后根据用户的查询将会获取一些文档，即检索结果。在把检索结果反馈给用户之前，还可以对检索结果按照一定的次序排序，以符合用户需要的文档能够排在更前面。

9.1.2　关于搜索引擎基础知识

搜索引擎可以通过各种查询方法来实现信息的查询，比如顺序查询方法和索引查询方法。何为顺序查询方法呢？何为索引查询方法呢？

所谓顺序查询方法是指当用户进行查询时，对文档集合不做任何形式的预处理，而直接在文档中进行字符串的简单匹配。虽然该方式简单、容易实现，但是当查找的文件大小超过一定数量级别时，该方式的效率就不能满足实际需要。

所谓索引查询方法是指当用户进行查询时，查询的对象为对文档集合创建的特殊数据结构，该数据结构就是索引。该种方式只针对文档的信息相对稳定的情况，因为当文档中的信息发生变化时，还必须对索引进行更新。

注意：*在具体实现时，通常是定期更新索引，同时把索引添加到原来的索引里。*

当使用索引查询方式来实现搜索引擎时，首先需要对文档集合进行预处理，即建立索引结构。进行预处理的技术主要有 3 种：倒排索引、后缀数组和签名文件。其中后缀数组技术虽然在短语查询中具有很快的速度，但是在具体构造和维护时都很复杂；签名文件技术虽然在 20 世纪 80 年代时期比较流行，但是随着时代的发展逐步被倒排索引技术取代。Lucene 全文搜索组件就是利用该技术实现对信息排序，该技术对关键词的搜索非常有效。

倒排索引是针对单词的索引结构，其结构由“词典”和“出现情况”两部分组成，使用的是“词到文档”的映射关系。为什么要叫倒排而不是正排呢？

如果为正排索引则应该使用“文档到词”的映射关系，其与具有“词到文档”映射关系的倒排索引相比，在“关键字”搜索方面具有很大的差距。为了便于讲解，下面将通过一个具体的实例来说明两种映射关系的差异。

现在存在两篇文档：文档 A 的内容为 We are chinese；文档 B 的内容为 chinese is a big country。如果建立的是正常的索引结构，其映射关系如表 9.1 所示。

表 9.1　正常索引的映射关系

文　档　号	出现的单词	出现的次数
A	chinese	1
B	chinese	1
A	We	1
…	…	…

如果建立的是倒排的索引结构，其映射关系如表 9.2 所示。

表 9.2　倒排索引的映射关系

出 现 单 词	出现的文档	出现的次数
chinese	AB	1，1
We	A	1
Are	A	1
…	…	…

在现实中用户通常用“关键字”进行查询，如果想查询 chinese 关键字，对于表 9.1 需要遍历所有的索引记录，对于表 9.2 却不需要遍历所有的索引记录。所以倒排索引更适合以关键字为标准建立索引。

9.2　网络蜘蛛（Web Spider）

网络蜘蛛英文名为 Web Spider，是一种专业的 Bot 程序。通过该名字，可以形象地想到 Web Spider 程序就是在互联网这个蜘蛛网上爬来爬去的蜘蛛。

9.2.1　关于网络蜘蛛的描述

网络蜘蛛主要实现什么功能呢？主要是在互联网这个大环境中获取所需要的网页，同时还可以通过扫描 Web 站点的主页来得到这个站点的文件清单和层次机构，从而判断出该网站中断的超链接和拼写错误等。

网络蜘蛛具体运行时，首先会从一个简单的 Web 页面（通常为首页）上开始执行，然后通过该网页中的链接访问其他页面，如此反复就可以访问该站点的所有页面。如果把整个互联网当成一个网站，那么理论上网络蜘蛛就可以用这个原理把互联网上所有的网页都抓取下来。

基于因特网的搜索引擎是 Web Spider 的最典型的应用。例如搜索巨头 Google 和 Baidu 公司就利用网络机器人程序来遍历 Web 站点，以创建并维护这些大型数据库。

网络蜘蛛通过链接从一个网页迁移到另一个网页来抓取网页，那么其是根据什么来决定抓取网页的先后顺序呢？可以采用 3 种策略：IP 地址搜索策略、广度优先策略和深度优先策略。

所谓 IP 地址搜索策略，首先网络蜘蛛会获得一个起始的 IP 地址，然后根据 IP 地址递增的方式搜索本 IP 地址段后的每一个 IP 地址中的网页，它完全不考虑各网页中的链接地址。虽然该策略可以搜索到一些没被其他网页引用的网页的信息源，但是却不适合大规模搜索。

所谓广度优先策略，首先网络蜘蛛会先抓取起始网页中链接的所有网页，然后再选择其中的一个链接网页，继续抓取在此网页中链接的所有网页。只有所有起始网页链接页面中的链接页面被抓取完，才会抓取下一层链接页面。该策略在具体实现时，可以使用队列

的数据结构，即当发现链接后并不调用自己本身而是把链接加入到等待队列中。当扫描完当前页面后会根据制定的策略访问队列中的下一个链接地址。

所谓深度优先策略，首先网络蜘蛛会从起始页开始，一个链接一个链接地跟踪下去，处理完这条线路上所有链接页面之后再转入起始页的下一个链接。该策略具体实现时，可以利用递归结构，即在一个方法中调用自己本身的程序设计技术。

当网络蜘蛛采用广度优先策略时，可以采用线程并行的技术，以提高抓取效率，但是在具体实现时比较复杂。当采取深度优先时，虽然可以利用递归结构来实现，但是耗费内存且不能使用多线程技术。

9.2.2　关于网络蜘蛛的基础知识

在具体实现网络蜘蛛时，虽然理论上可以抓取所有的网页，但是现实中却会忽略一些不太重要的网站网页，那如何决定访问层数呢？如果网络蜘蛛抓取的网页涉及加密数据和网页权限时，那又如何处理呢？如果想实现网络蜘蛛与所抓取网页的站点的交互，那又如何处理呢？

对于第 1 个问题，可以为网络蜘蛛设置访问的层数。例如，A 为起始网页，属于 0 层，B、C、D、E、F 网页属于第 1 层，G、H 网页属于第 2 层，I 网页属于第 3 层。如果网络蜘蛛设置的访问层数为 2 的话，网页 I 是不会被访问到的。使用该种方式，有时候会让网站上一部分网页能够在搜索引擎上搜索到，而另外一部分却不能被搜索到。

对于第 2 个问题，可以通过获取站点赋予的权限来实现对网页进行搜索。通过权限可以使搜索更人性化：例如对于一些出售报告的网站，他们希望搜索引擎能搜索到他们的报告，但又不能完全让搜索者查看，这样就需要给网络蜘蛛提供相应的用户名和密码。网络蜘蛛可以通过所给的权限对这些网页进行网页抓取，从而提供搜索。而当搜索者点击查看该网页的时候，同样需要搜索者提供相应的权限验证。

对于第 3 个问题，由于网络蜘蛛抓取网页时不同于一般的网页访问，所以需要一些特殊的手段来实现两者的交互。由于每个网络蜘蛛都有自己的名字，所以在抓取网页的时候一般都会向网站标明自己的身份。网络蜘蛛在抓取网页的时候会发送一个请求，这个请求中就有一个字段为 User－agent，用于标识此网络蜘蛛的身份。例如 Google 网络蜘蛛的标识为 GoogleBot，Baidu 网络蜘蛛的标识为 BaiDuSpider，Yahoo 网络蜘蛛的标识为 Inktomi Slurp。网站管理员只要查看网站上的日记记录就可以知道那些搜索引擎的网络蜘蛛过来过。

网络蜘蛛进入一个网站，一般会访问一个特殊的文本文件 Robots.txt，这个文件一般放在网站服务器的根目录下。网站管理员可以通过 Robots.txt 来决定哪些目录网络蜘蛛不能访问，或者哪些目录对于某些特定的网络蜘蛛能访问。例如有些网站的可执行文件目录和临时文件目录不希望被搜索引擎搜索到，那么网站管理员就可以把这些目录定义为拒绝访问目录。

注意： 由于 Robots.txt 只是一个协议，所以如果网络蜘蛛不遵循这个协议，那么网站管理员也无法阻止网络蜘蛛对于某些页面的访问。

9.2.3　如何创建 Web Spider 程序

通过前面内容的学习，大家已经知道了什么是 Web Spider，那么如何创建关于 Web Spider 的程序呢？由于 Web Spider 程序属于 Bot 程序，所以在具体开发 Web Spider 程序时，需要引入 Bot.jar 文件。

Spider.java 实现了 Web Spider 的创建，主要内容如代码 9.1 所示。

代码 9.1　创建网络蜘蛛：Spider.java

```
...
public class Searcher
    implements ISpiderReportable {
  public static void main(String[] args) throws Exception {
...
  public boolean foundInternalLink(String url) {    //发现内部连接时调用
    return false;
  }
  public boolean foundExternalLink(String url) {    //发现外部连接时调用
    return false;
  }
  public boolean foundOtherLink(String url) {       //发现其他连接时调用
    return false;
  }
  public void processPage(HTTP http) {              //处理被请求的网页
    System.out.println("扫描网页：" + http.getURL());
    new HTMLParse(http).start();
  }
  public void completePage(HTTP http, boolean error) {//处理一个被处理的网页
  }
  public boolean getRemoveQuery() {                 //处理查询字符串
    return true;
  }
  public void spiderComplete() {                    //完成工作后调用的方法
  }
}
```

【代码解析】

在上述代码中，继承了 IspiderReportable()方法，查看帮助文档可以发现该方法的定义如下：

```
public interface IspiderReportable{
     public boolean foundInternalLink(String url);
     public boolean foundExternalLink(String url);
     public boolean foundOtherLink(String url);
     public void processPage(HTTP page);
     public void completePage(HTTP page,boolean error);
     public boolean getRemoveQuery();
     public void SpiderComplete();
}
```

- 当发现内部连接时调用 foundInternalLink()方法，其中参数 url 表示程序发现的 URL，如果返回的结果为 true，则会加入作业中，否则就不加入。

- 当发现外部连接时调用 foundExternalLink()方法，其中参数 url 表示程序发现的 URL，如果返回的结果为 true，则会加入作业中，否则就不加入。
- 当发现其他连接时调用 foundOtherLink()方法，其中参数 url 表示程序发现的 URL，即非 HTML 网页的地址，例如 E-mail 或者 FTP 等。如果返回的结果为 true，则会加入作业中，否则就不加入。
- 当具体处理网页时，就会调用 processPage()方法。如果需要请求一个被处理的网页，则会调用 completePage()方法。如果想确定查询字符串是否应删除，可以调用 getRemoveQuery()方法。最后，当没有其他的工作时，则会调用 SpiderComplete()方法。

9.3　下载和分析 Lucene 全文搜索组件

Lucene 全文搜索组件是 Jakarta Apache 的开源项目，由资深全文索引/检索专家 Doug Cutting 贡献，主要解决各种中小型应用程序加入全文检索功能。其实 Lucene 全文搜索组件只是一个用 Java 编写的全文搜索引擎工具包。

9.3.1　下载 Lucene 全文搜索组件

自从 Lucene 全文搜索组件问世之后，程序员们不仅使用其构建具体的全文检索应用，而且还将其集成到各种系统软件，甚至某些商业软件也采用了 Lucene 全文搜索组件作为其内部全文检索子系统的核心。该组件具体的下载步骤如下。

（1）首先访问下载 Lucene 全文搜索组件的官方网站（http://www.apache.org/），如图 9.1 所示。在该页面中选择 Lucene 链接，就会转入关于 Lucene 全文搜索组件的页面。

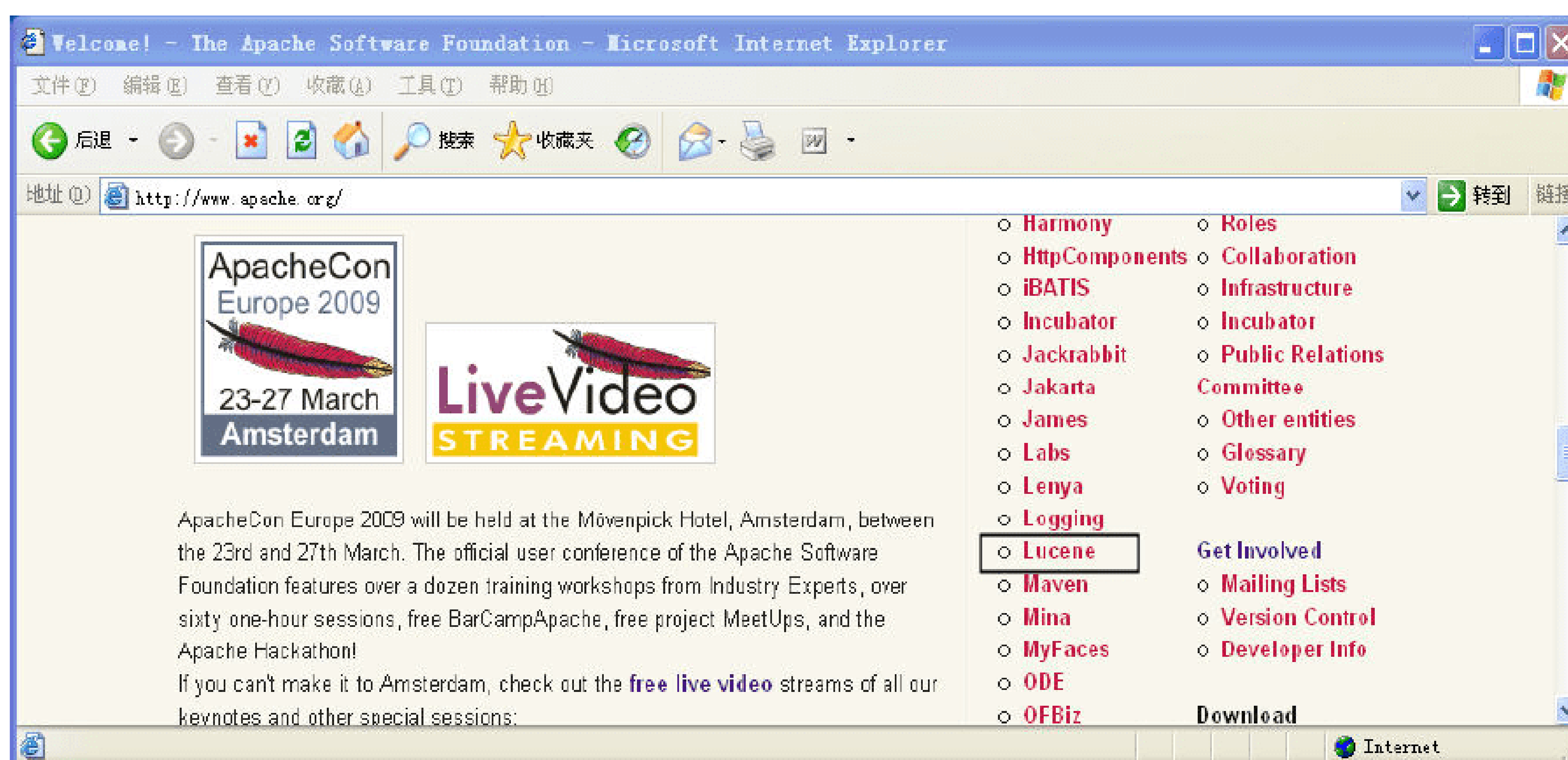


图 9.1　Lucene 全文搜索组件首页

（2）在关于 Lucene 全文搜索组件的页面中，单击 java 导航栏就会进入关于 Java 语言编程 Lucene 全文搜索组件的页面（如图 9.2 所示）。在该页面中单击 free download 链接就可以转入下载 Lucene 全文搜索组件的页面。

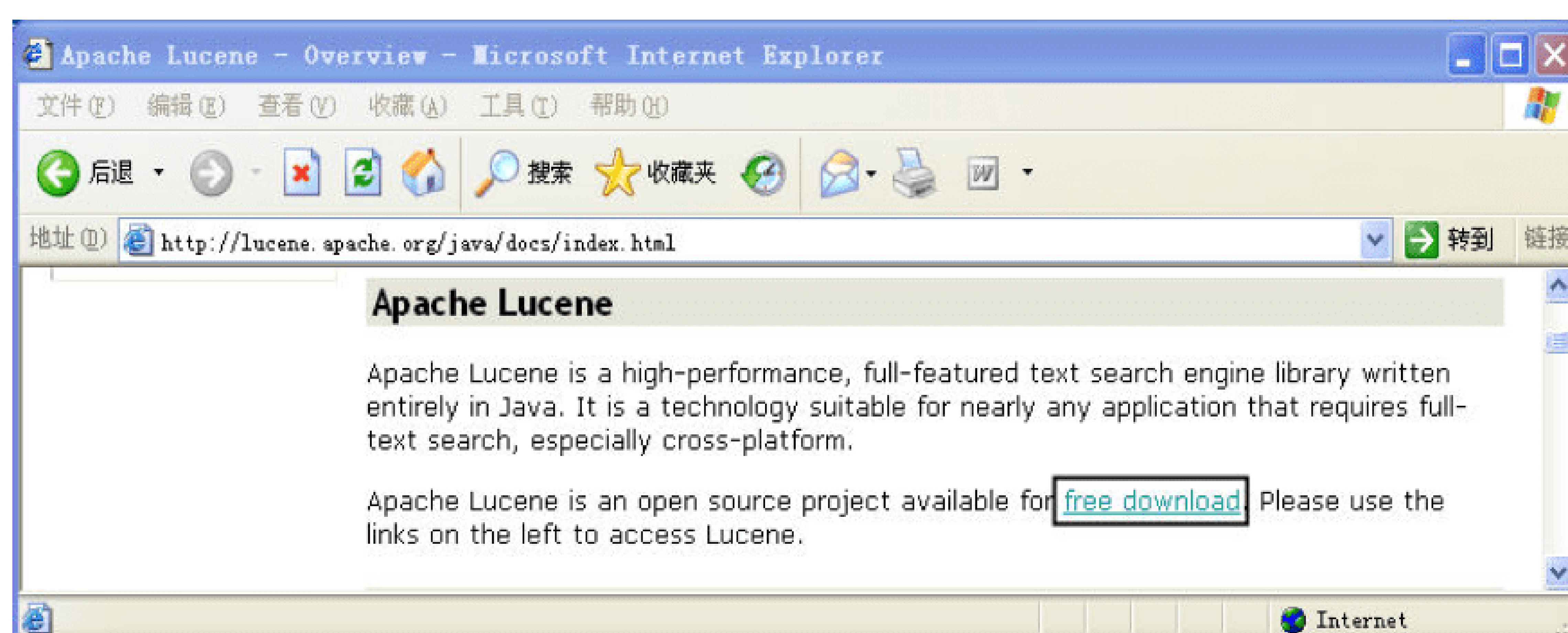


图 9.2　下载 Lucene 全文搜索组件页面

（3）在关于下载 Lucene 全文搜索组件的页面中（如图 9.3 所示），单击 http://apacche.freelamp.com/lucene/java/链接就可以进入下载 Lucene 全文搜索组件的真正页面。在该页面单击 lucene-2.6.1.zip 链接就可以实现该组件的下载，如图 9.4 所示。

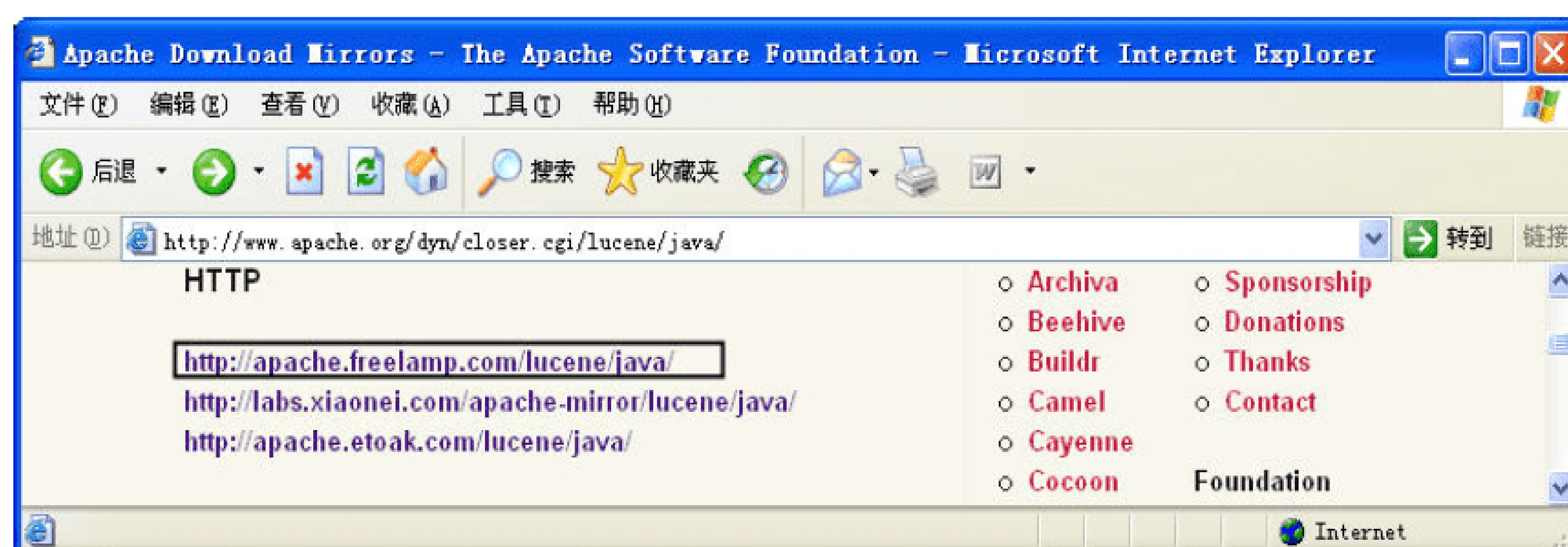


图 9.3　选择 Lucene 全文搜索组件下载地址

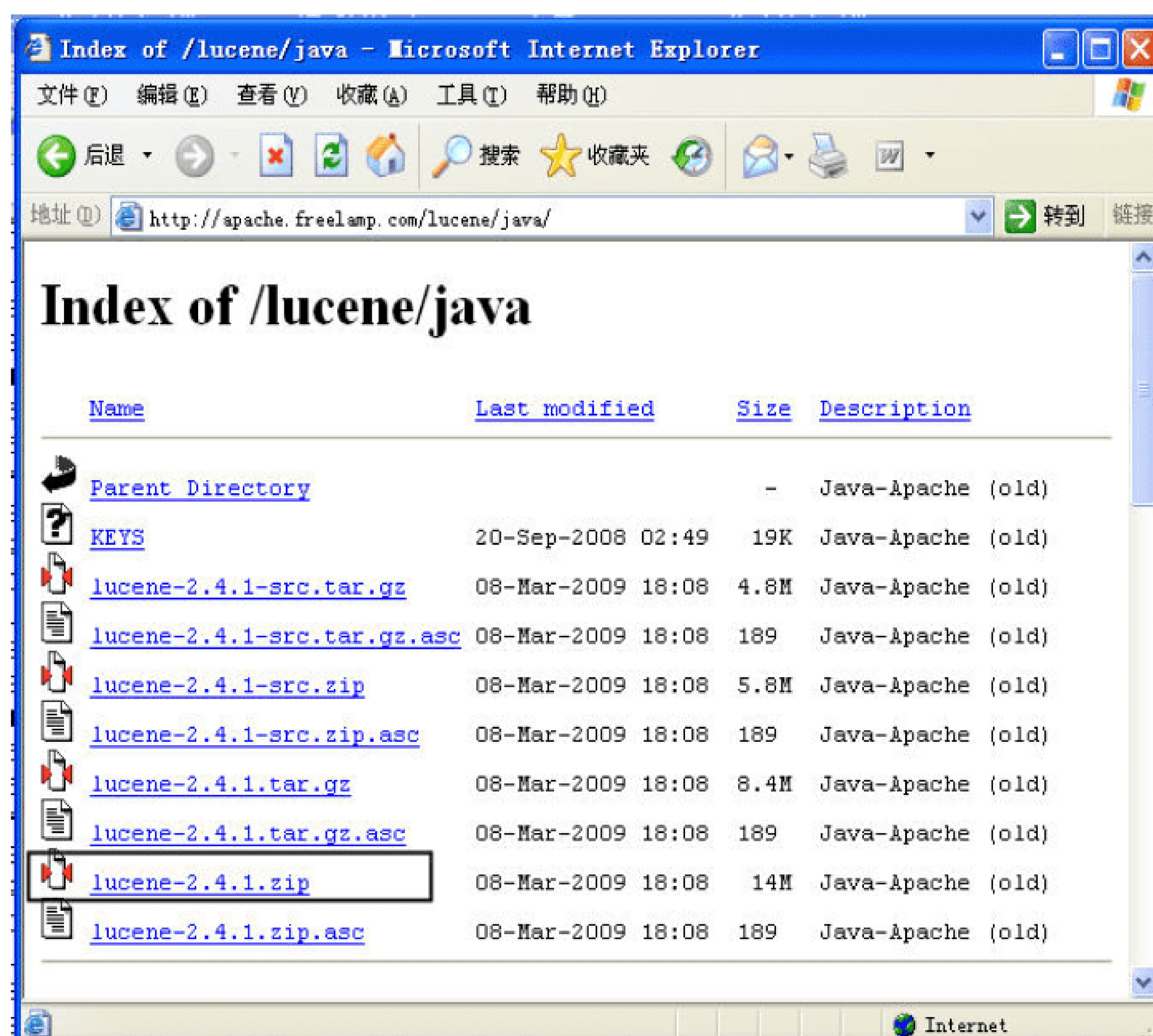


图 9.4　Lucene 全文搜索组件下载

至此，就完成了对 Lucene 全文搜索组件的下载。

9.3.2　Lucene 全文搜索组件目录简介和开发文档

9.3.1 节介绍了如何下载 Lucene 全文搜索组件，下载完该组件后就可以在 Java Web 项目中使用该组件。在具体使用之前，先解压 Lucene-2.4.1.zip 文件，该压缩包目录如图 9.5 所示。

在上述目录中，各个目录的具体含义如下。

- contrib 文件：该文件是 Lucene 全文搜索组件的一些插件。
- docs 文件：该文件是 Lucene 全文搜索组件的帮助文档。
- src 文件：该文件是 Lucene 全文搜索组件的源代码。
- lucene-core-2.6.1.jar 和 lucene-demos-2.6.1.jar：这两个 jar 文件是 Lucene 全文搜索组件的核心 jar 文件。
- luceneweb.war：该 war 文件为 Lucene 全文搜索组件的演示项目。

只需要通过 lucene-core-2.6.1.jar 文件就可以在 Java Web 项目中实现搜索引擎功能。解压 lucene-core-2.6.1.jar 文件，该压缩包目录如图 9.6 所示。

名称	大小	类型
contrib		文件夹
docs		文件夹
src		文件夹
BUILD. txt	4 KB	文本文档
build. xml	10 KB	XML 文档
CHANGES. txt	123 KB	文本文档
LICENSE. txt	13 KB	文本文档
lucene-core-2. 4. 1. jar	804 KB	Executable
lucene-demos-2. 4. 1. jar	55 KB	Executable
luceneweb. war	799 KB	WAR 文件
NOTICE. txt	1 KB	文本文档
README. txt	2 KB	文本文档

图 9.5　目录结构

```
lucene-core-2. 4. 1. jar
  org. apache. lucene
  org. apache. lucene. analysis
  org. apache. lucene. analysis. standard
  org. apache. lucene. document
  org. apache. lucene. index
  org. apache. lucene. queryParser
  org. apache. lucene. search
  org. apache. lucene. search. function
  org. apache. lucene. search. payloads
  org. apache. lucene. search. spans
  org. apache. lucene. store
  org. apache. lucene. util
  org. apache. lucene. util. cache
  META-INF
    LICENSE. txt
    MANIFEST. MF
    NOTICE. txt
```

图 9.6　目录结构

在上述目录结构中，各个 jar 文件的含义如下。

- analysis：该包主要用于向 Lucene 全文搜索提供对文本进行分词、过滤等操作的支持。
- standard：该包主要用于向 Lucene 全文搜索提供标准分析器。
- document：该包主要用于向 Lucene 全文搜索提供对 Document 和 Field 的各种操作的支持。
- index：该包主要用于向 Lucene 全文搜索提供建立索引时的各种操作。
- util：该包主要用于向 Lucene 全文搜索提供常用工具类和常量的支持。
- store：该包主要用于向 Lucene 全文搜索提供对索引存储的支持。
- search：该包主要用于向 Lucene 全文搜索提供检索的功能。
- queryParser：该包主要用于向 Lucene 全文搜索提供索引时的分析支持。

为了便于使用，可以复制 lucene-core-2.6.1.jar 这个 jar 文件到相应的文件夹中，然后把这个 jar 文件在 MyEclipse 开发环境中专门建立一个名为 lucene 的用户库，具体步骤如下。

（1）首先通过菜单 MyEclipse | Preferences 打开属性对话框，在该对话框的左边项目中选择 Java | Build Path | User Libraries 目录打开配置用户库的对话框，如图 9.7 所示。

（2）在配置用户库对话框中首先通过单击 New 按钮创建一个名为 lucene 的用户库，如图 9.8 所示。接着单击名为 lucene 的用户库，然后通过单击 Add JARs 按钮为该用户库添加相应的 jar 包，如图 9.9 所示。当配置完该用户库后，只需要单击 OK 按钮就可以完成对该用户库的配置。

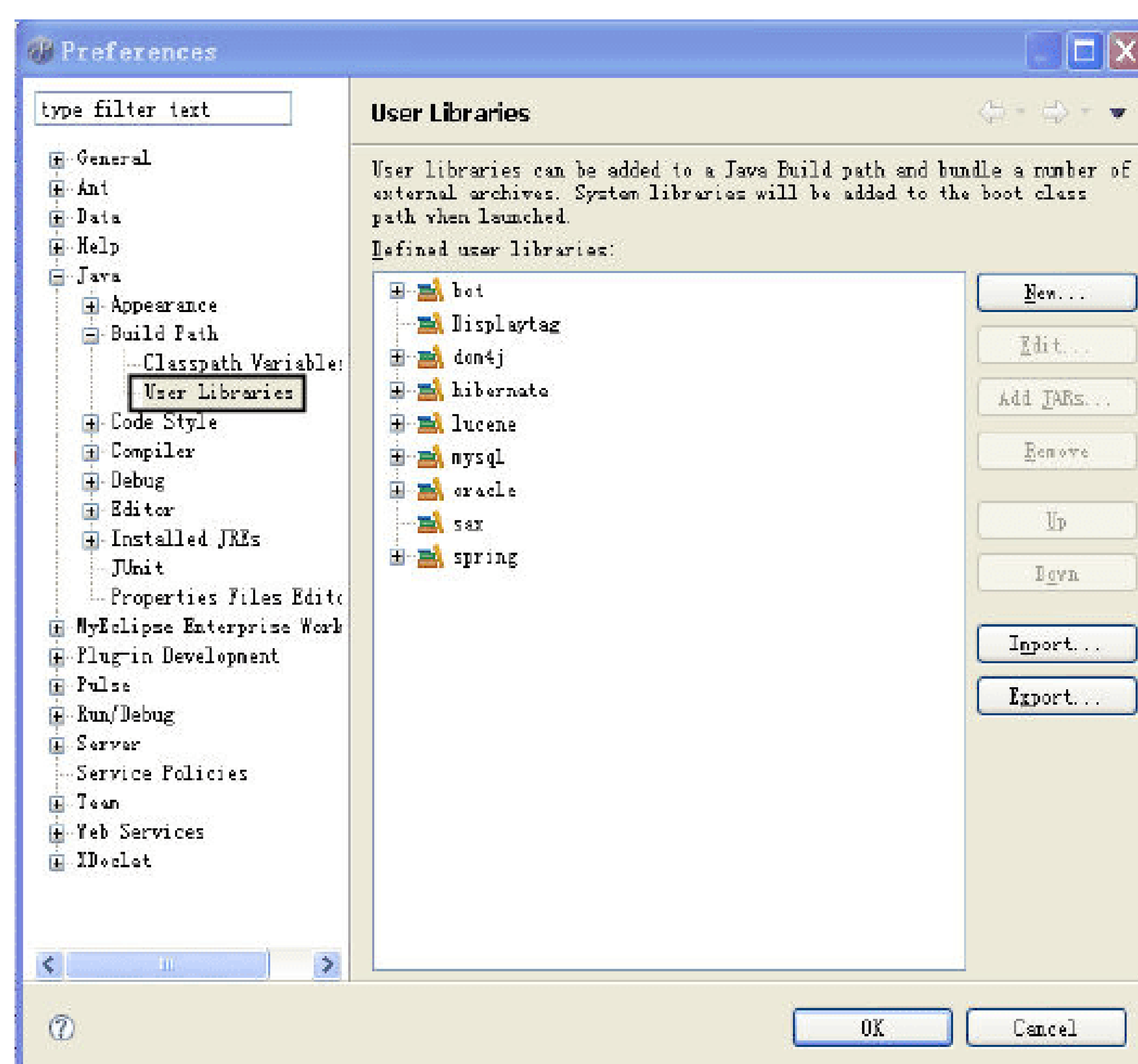


图 9.7　配置用户库对话框

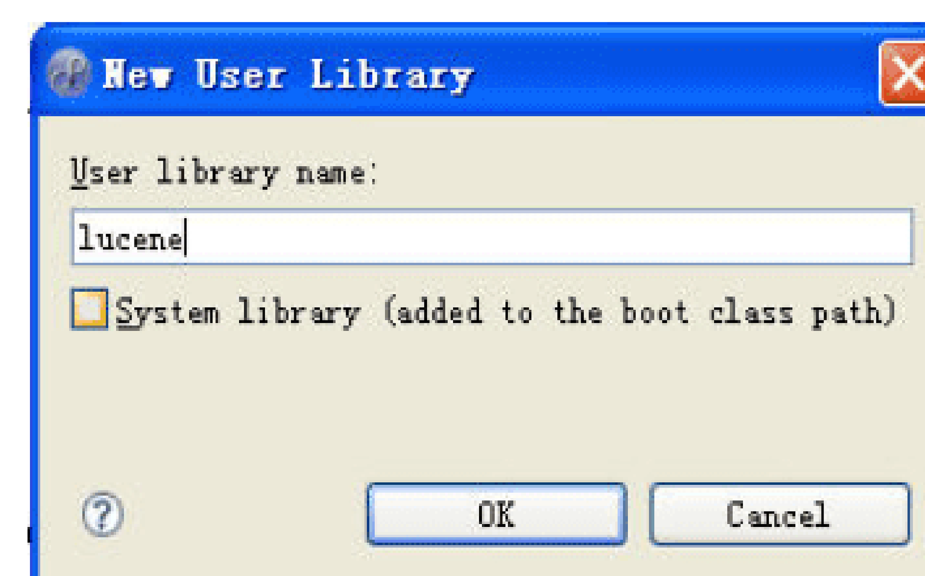


图 9.8　创建 lucene 用户库

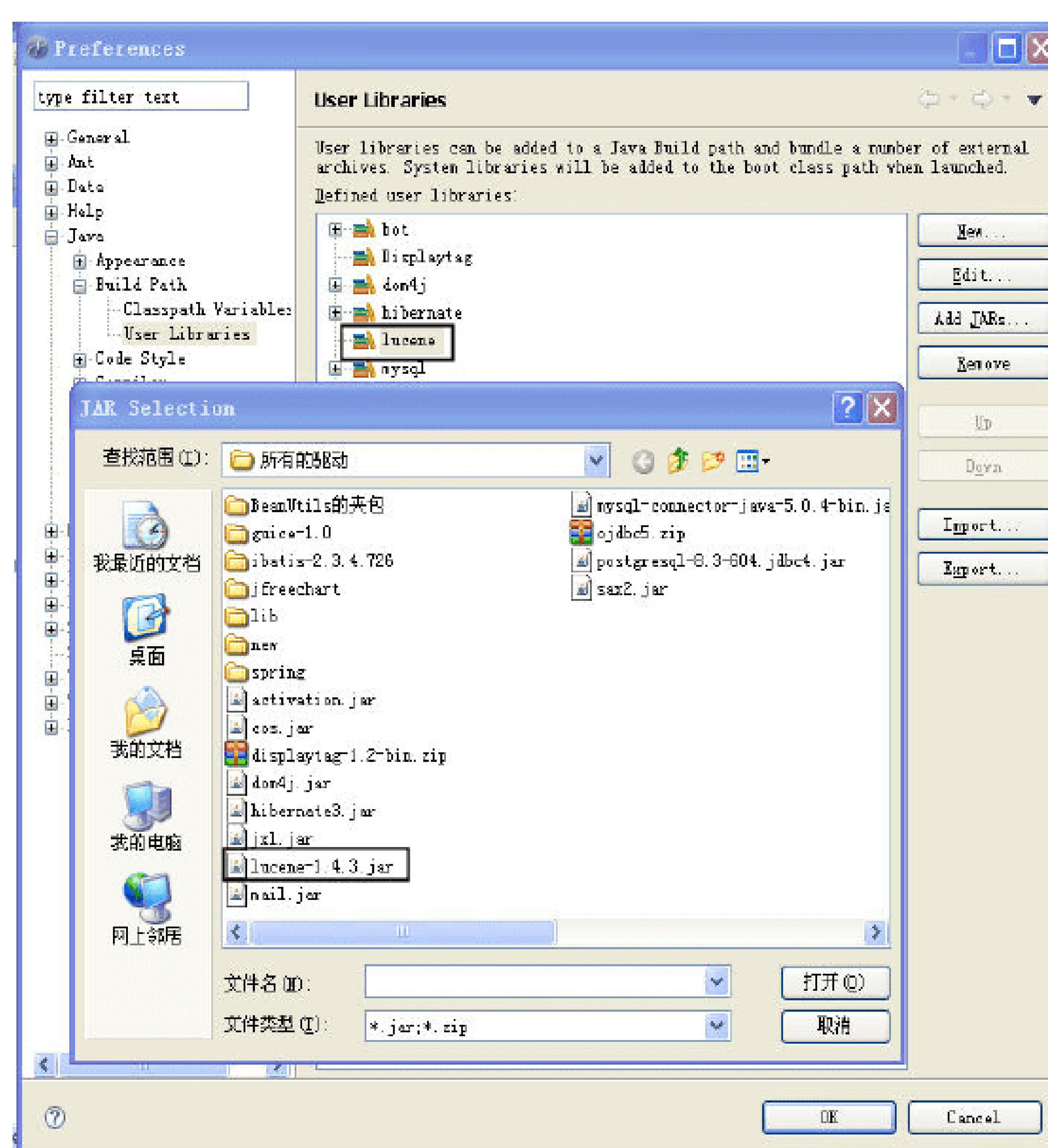


图 9.9　配置 lucene 用户库

（3）当通过 Add JARs 按钮为 lucene 用户库添加完 jar 包时，具体的 jar 包列表如图 9.10 所示。

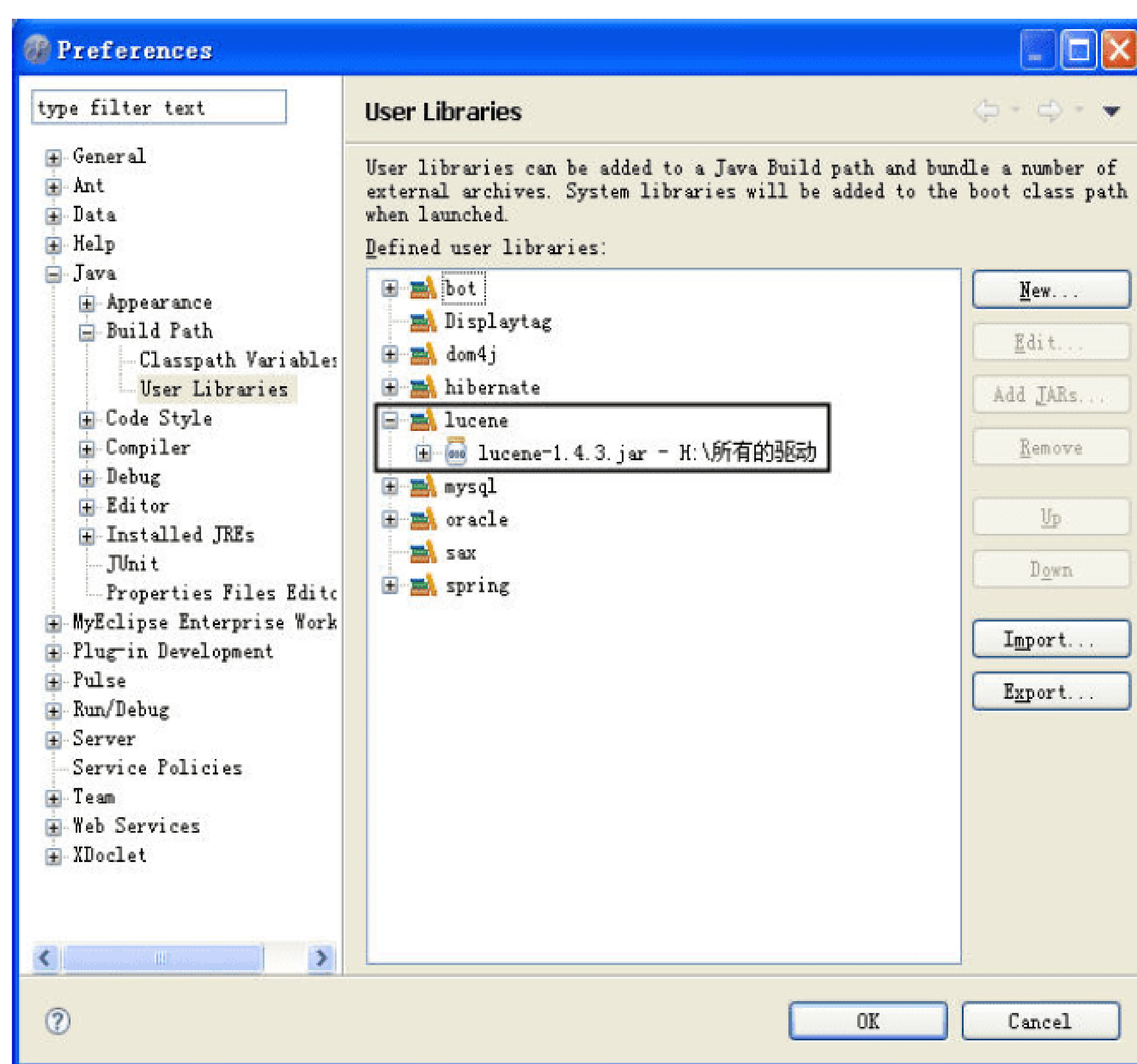


图 9.10　相关 jar 包

至此，就完成了关于 lucene 用户库的配置。

9.4　初步使用 Lucene 全文搜索组件

通过上述章节的学习已经知道了 Lucene 全文搜索组件的一些基本知识，本章将通过一些简单的事例来讲解如何使用 Lucene 全文搜索组件。如果想使用 Lucene 全文搜索组件，首先需要为文件创建索引，接着才能实现查找功能。

9.4.1　创建索引

由于 Lucene 全文搜索组件是通过索引查询方式来实现搜索引擎，所以通过 Lucene 全文搜索组件实现搜索引擎首先需要建立索引。为搜索引擎建立索引是一门高深技术，涉及的知识面比较广，本节只是简单介绍一下如何创建索引。

利用 Lucene 全文搜索组件建立索引需要经过 3 个步骤。

（1）获取需要进行索引的文件和创建存放索引文件。

（2）为需要进行索引的文件创建索引。

（3）把索引存储到索引文件中。

下面将通过一个简单的例子来讲解，利用 Lucene 全文搜索组件如何实现索引文件的创建。代码 9.2 实现对本地计算机上存在的 4 个文件创建索引文件。

代码 9.2　建立索引：LuceneIndexTest.java

```
...
public class LuceneIndexTest {
```

```
    public static void main(String[] args) throws Exception {  //主函数
        LuceneIndexTest indexer = new LuceneIndexTest();
                                                //获取 LuceneIndexTest 对象
        indexer.writeToIndex();                 //调用 writeToIndex()方法
        indexer.close();                        //调用 close()方法
    }
    private IndexWriter writer = null;          //创建索引器
    public LuceneIndexTest() {                  //创建构造函数
        try {
            writer = new IndexWriter(Constants.INDEX STORE PATH,
                    new StandardAnalyzer(), true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 将要建立索引的文件构造成一个 Document 对象
    private Document getDocument(File f) throws Exception {
        Document doc = new Document();
        FileInputStream is = new FileInputStream(f);
        Reader reader = new BufferedReader(new InputStreamReader(is));
        doc.add(Field.Text("contents", reader));
        doc.add(Field.Keyword("path", f.getAbsolutePath()));
        return doc;
    }
    public void writeToIndex() throws Exception {  //把文档添加到索引中
        File folder = new File(Constants.INDEX FILE PATH);
        if (folder.isDirectory()) {
            String[] files = folder.list();
            for (int i = 0; i < files.length; i++) {
                File file = new File(folder, files[i]);
                Document doc = getDocument(file);
                System.out.println("正在建立索引 : " + file + "");
                writer.addDocument(doc);
            }
        }
    }
    public void close() throws Exception {          //实现索引文件的存储
        writer.close();
    }
}
```

运行该程序就会出现如图 9.11 所示的结果。

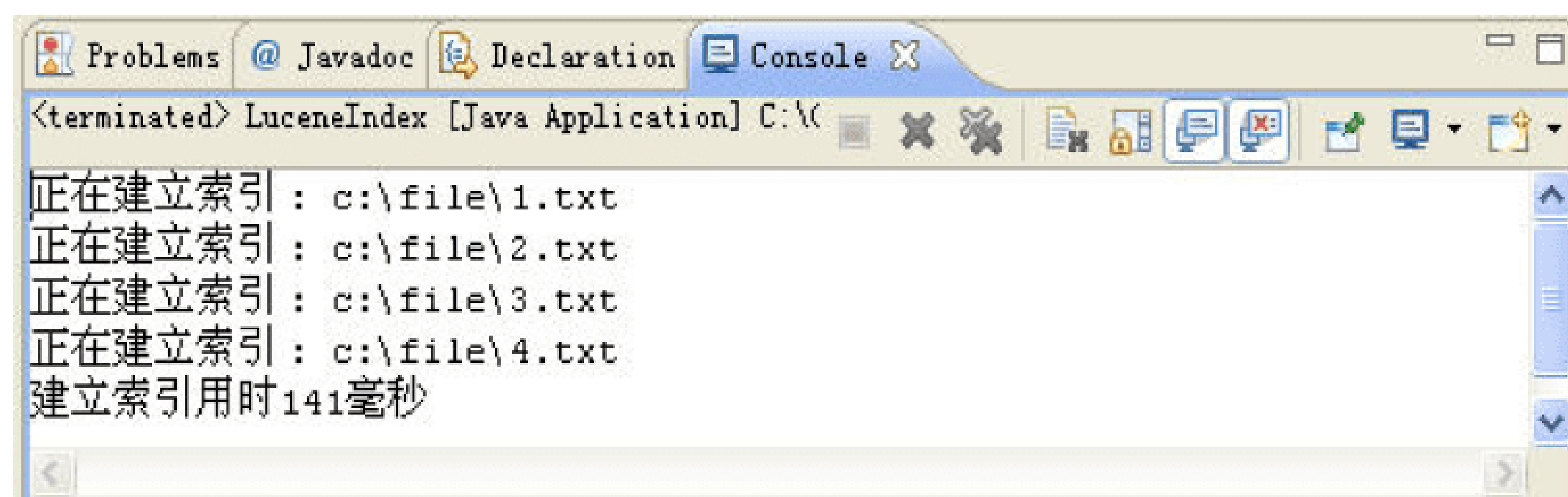


图 9.11　运行结果

【代码解析】

- ❑ 首先通过 indexWeriter 类创建索引书写器，在具体创建 indexWeriter 类对象时，需要 3 个参数，其中主要的一个就是表示存放索引路径上的参数。由上述代码的构

造方法实现该功能。

- 获取到 indexWeriter 类对象后，就可以通过该类的 addDocument()方法把所需要索引的文档添加到索引，在上述代码中 writeToIndex()方法实现该功能。由于 addDocument()方法的参数为 Document 类型，所以需要把所需要索引的文档转换成 Document 类型。该功能由上述 getDocument(File f)方法实现。
- 如果想把索引文件存储起来，必须调用 indexWeriter 类的 close()方法，由上述代码的 close()方法实现该功能。

在上述代码的 writeToIndex()方法和 LuceneIndexTest()方法中，分别出现了 Constants 类，该类主要通过常量定义了所需要进行索引的文件和存放索引文件的路径，具体内容如代码 9.3 所示。

代码 9.3　定义相关路径：Constants.java

```
...
public class Constants {
    public final static String INDEX_FILE_PATH = "c:\\file";
    public final static String INDEX_STORE_PATH = "c:\\index";
}
```

打开存放索引的文件，就可以看到自动创建的索引文件，如图 9.12 所示，而存放所要创建索引文件如图 9.13 所示。

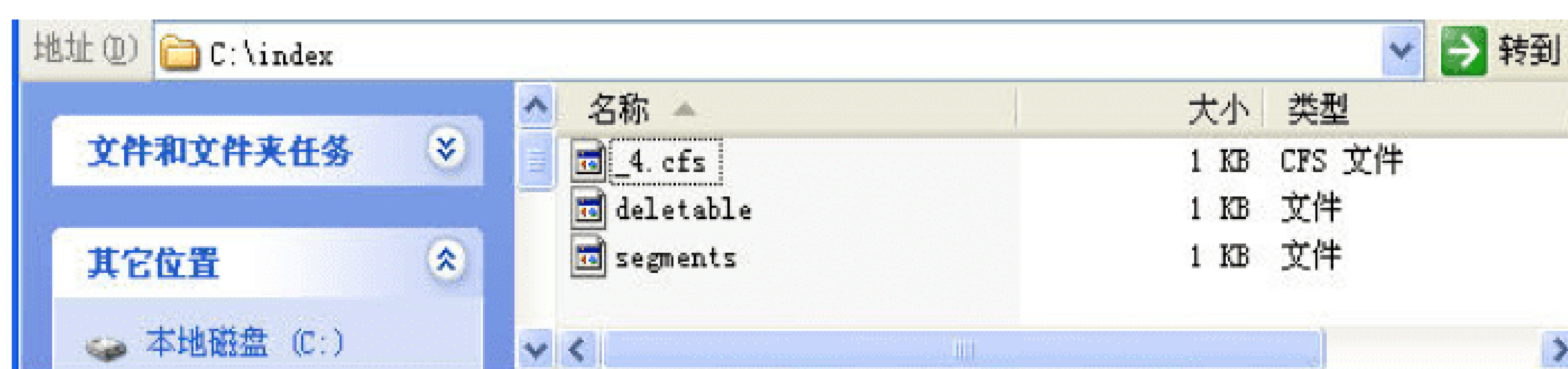


图 9.12　存放索引的文件

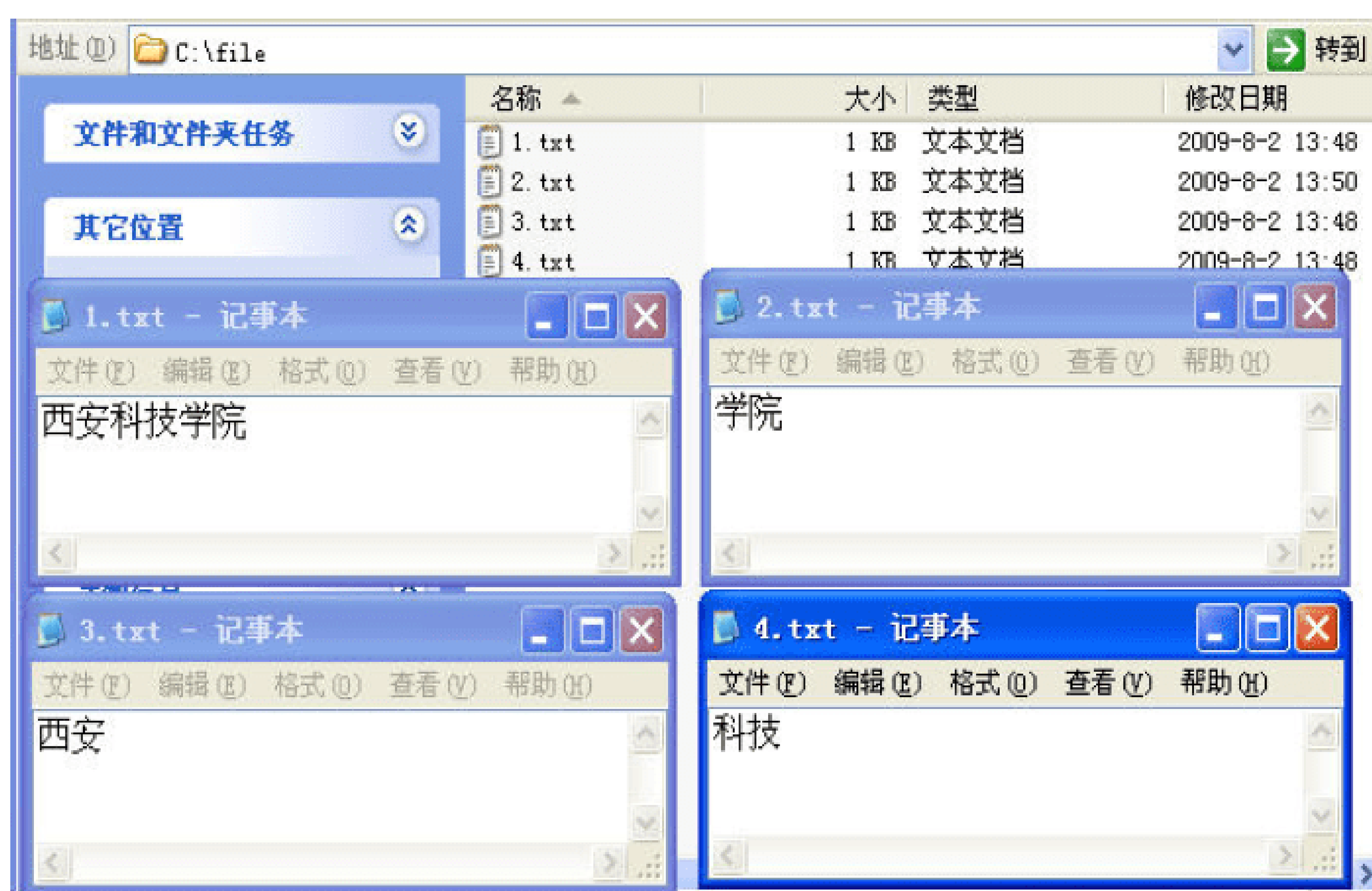


图 9.13　需要创建索引的文件

9.4.2　实现搜索

利用 Lucene 全文搜索组件创建搜索引擎最重要的就是建立索引和实现搜索，其中索引

的创建是实现搜索的基础条件。在实现搜索功能时，要考虑多方面的内容。即搜索过程的响应性、搜索结果的准确性。Lucene 全文搜索组件之所以成为一个非常优秀工具包，是因为其拥有高效的搜索算法和对搜索结果丰富的排序方法。

为搜索引擎实现搜索是一门高深技术，涉及的知识面也比较广，本节只是简单地介绍 Lucene 全文搜索组件实现搜索的一般流程。LuceneSearchTest.java 实现简单的搜索功能，具体内容如代码 9.4 所示。

代码 9.4　实现简单的搜索：LuceneSearchTest.java

```
...
public class LuceneSearchTest {
    public static void main(String[] args) throws Exception {
        LuceneSearchTest test = new LuceneSearchTest();
                                            //创建 LuceneSearch 类型对象
        Hits h = null;                      //创建 Hits 对象
        h = test.search("西安");            //查询字符串
        test.printResult(h);                //输出查询结果
        h = test.search("学院");            //查询字符串
        test.printResult(h);                //输出查询结果
        h = test.search("科技");            //查询字符串
        test.printResult(h);                //输出查询结果
    }
    public LuceneSearchTest() {             //构造函数
        try {
            searcher = new IndexSearcher(IndexReader
                    .open(Constants.INDEX STORE PATH));
                                            //创建 IndexSearcher 类型对象
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private IndexSearcher searcher = null;
    private Query query = null;
    public final Hits search(String keyword) {      //实现搜索功能
        System.out.println("正在检索关键字 : " + keyword);
        try {
            query = QueryParser.parse(keyword, "contents",
                                            //将关键字包装成 Query 对象
                    new StandardAnalyzer());
            Hits hits = searcher.search(query);
            return hits;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    public void printResult(Hits h) {       //输出搜索结果
        if (h.length() == 0) {
            System.out.println("对不起，没有找到您要的结果。");
        }
        else
        {
            for (int i = 0; i < h.length(); i++) {
                try {
                    Document doc = h.doc(i);
```

```
                    System.out.print("这是第" + i + "个检索到的结果，文件名
                    为：");
                    System.out.println(doc.get("path"));
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
        System.out.println("--------------------------");
    }
}
```

运行该程序就会出现如图 9.14 所示的结果。

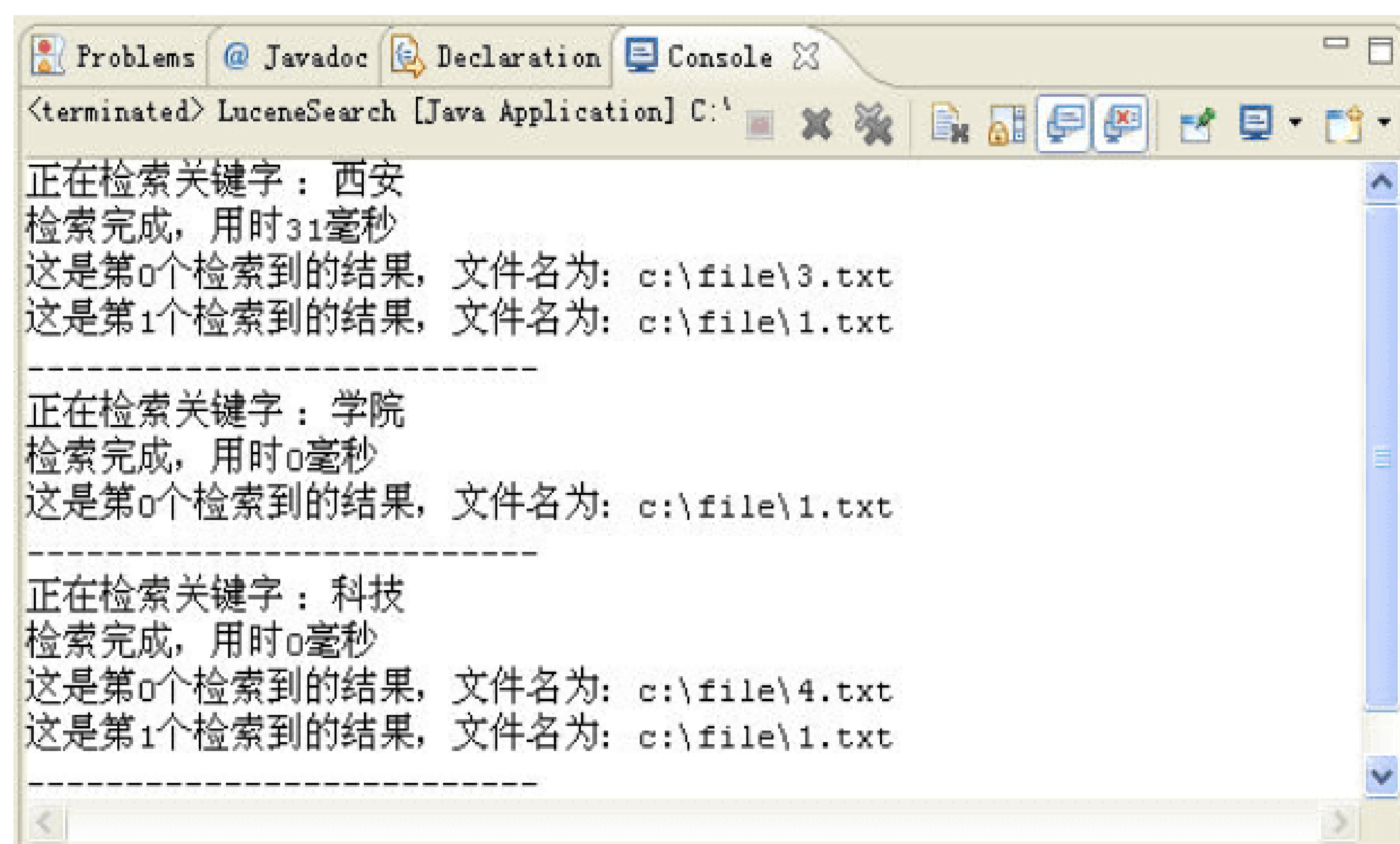


图 9.14　运行结果

【代码解析】

- 在上述代码中，printResult()方法实现输出查询结果，由于 search()方法的返回结果为 Hits 类型，所以还需要把结果由 Hits 类型转换成 Document 类型，再输出结果的相关信息。
- 该代码如果想运行成功，同样也需要调用定义文件路径的 Constants 类。
- 上述代码中的 search()方法实现查询功能，在该方法中主要通过如下代码：

```
Hits hits = searcher.search(query);
```

实现查询功能。由于 IndexSearcher 类的 search()方法的参数为 Query 类型，所以必须把要查询的内容 keyword 转换成 Query 类型。通过如下的代码：

```
query = QueryParser.parse(keyword, "contents",
        new StandardAnalyzer());
```

实现查询内容的包装。上述一句代码可以分解成如下代码：

```
QueryParser parser=new QueryParser("contents",new StandardAnalyzer());
query= parser. parse(keyword);
```

9.5　新闻搜索引擎具体实现

新闻搜索引擎是从指定的 Web 页面中按照超链接进行解析、搜索，并把搜索到的每条

新闻进行索引后存储到数据库。然后通过 Web 服务器接受客户端请求后从索引数据库中搜索出所匹配的新闻。

为了便于讲解，本章的新闻搜索引擎只解析本地服务器中的一个页面，同时也没有把关于解析页面的索引存放到数据库中，而是存储到本地服务器的文件中。该系统涉及的知识除了 Lucene 全文搜索组件，还涉及 Web Spider 知识。

9.5.1　为新闻搜索引擎创建索引工具类

在为新闻搜索引擎内容创建索引时，引入了一个工具类 Index，该类用来实现对内容创建索引，具体内容如代码 9.5 所示。

代码 9.5　创建索引工具类：Index.java

```
public class Index {
  IndexWriter  _writer = null;                          //创建 IndexWriter 变量
  Index() throws Exception {                            //构造函数
    _writer = new IndexWriter("c:\\News\\index",
                          new ChineseAnalyzer(), true);
  }
  void AddNews(String url, String title) throws Exception {
                                                        //把每条新闻加入索引中
    Document  _doc = new Document();                    //创建 Document 对象
    _doc.add(Field.Text("title", title));               //设置字段
    _doc.add(Field.UnIndexed("url", url));
    _writer.addDocument( _doc);
  }
  void close() throws Exception {                       //优化并且清理资源
    _writer.optimize();
    _writer.close();
  }
}
```

注意：在上述代码中，AddNews()方法中的参数 url 表示新闻的 URL 地址，而 title 表示新闻的标题。

HTMLParse.java 类不仅通过解析 HTML 获取内容，而且还通过调用 Index 工具类实现对解析到的内容创建索引，该类的具体内容如代码 9.6 所示。

代码 9.6　对搜索的内容进行索引：HTMLParse.java

```
...
public class HTMLParse {
  HTTP _http = null;                                  //定义 HTTP 类型变量
  public HTMLParse(HTTP http) {                       //构造函数
    _http = http;
  }
  public void start() {                               //对 Web 页面进行解析后建立索引
    try {
      HTMLPage  _page = new HTMLPage( _http);
       _page.open( _http.getURL(), null);
      Vector  _links =  _page.getLinks();
      Index _index = new Index();
```

```
     Iterator  it =  links.iterator();
     int n = 0;
     while (_it.hasNext()) {
       Link  link = (Link)  it.next();
       String  herf = input( link.getHREF().trim());
       String _title = input(_link.getPrompt().trim());
        index.AddNews( herf,  title);
       n++;
     }
     System.out.println("共扫描到" + n + "条新闻");
     _index.close();
   }
   catch (Exception ex) {
     System.out.println(ex);
   }
 }
 public static String input(String str) {         //解决 Java 中的中文问题
   String temp = null;
   if (str != null) {
     try {
       temp = new String(str.getBytes("ISO8869 1"));
     }
     catch (Exception e) {
     }
   }
   return temp;
 }
}
```

注意：在上述代码中，input()方法中的参数 str 表示输入的中文。通过上述两个类就可以实现对新闻搜索引擎内容创建索引。

TestIndex.java 类用来测试上面创建索引是否成功，具体内容如代码 9.7 所示。

代码 9.7　实现查找功能：TestIndex.java

```
...
//引入 bot 相关 jar 包
import com.heaton.bot.HTTP;
import com.heaton.bot.HTTPSocket;
import com.heaton.bot.ISpiderReportable;
import com.heaton.bot.IWorkloadStorable;
import com.heaton.bot.Spider;
import com.heaton.bot.SpiderInternalWorkload;

public class TestIndex implements ISpiderReportable { //主函数
    public static void main(String[] args) throws Exception {
        IWorkloadStorable wl = new SpiderInternalWorkload();
        TestIndex  searcher = new TestIndex();
        Spider _spider = new Spider(_searcher,
                "http://www.chenshen.com/index.html", new HTTPSocket(),
                100, wl);
        _spider.setMaxBody(100);
         spider.start();
    }

    public boolean foundInternalLink(String url) {     //发现内部连接时调用
        return false;
```

```
    }
    public boolean foundExternalLink(String url) {     //发现外部连接时调用
        return false;
    }
    public boolean foundOtherLink(String url) {        //发现其他连接时调用
        return false;
    }
    public void processPage(HTTP http) {               //处理被请求的网页
        System.out.println("扫描网页: " + http.getURL());
        new HTMLParse(http).start();
    }
    public void completePage(HTTP http, boolean error) {
                                                    //处理一个被处理的网页
    }
    public boolean getRemoveQuery() {               //处理查询字符串
        return true;
    }
    public void spiderComplete() {                  //完成工作后调用的方法
    }
}
```

运行该程序就会出现如图 9.15 所示的结果。打开“C:\News\index”文件夹，索引文件如图 9.16 所示。

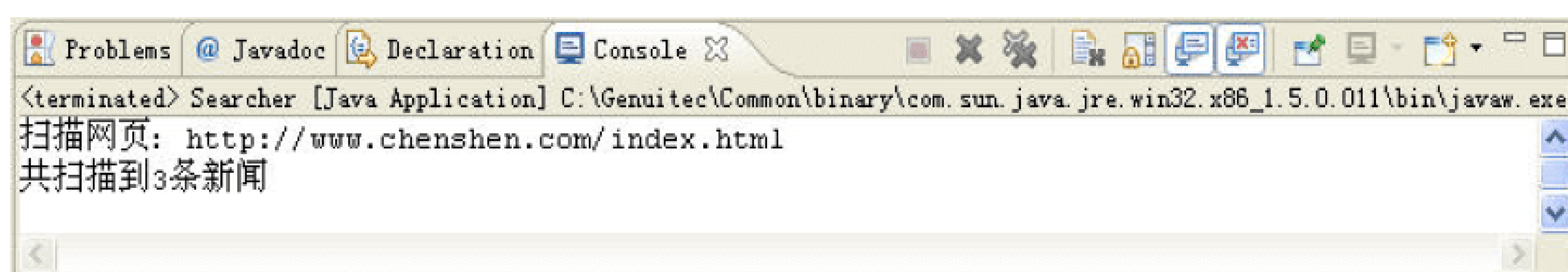


图 9.15　运行结果

9.5.2　为新闻搜索引擎实现搜索功能

当客户端发出请求后，服务器端首先通过 get()方法获取请求中的查询条件，然后调用相关搜索的开发包进行搜索操作，最后把搜索的结果以 HTTP 消息包的形式发送至客户端，从而完成一次搜索操作，具体流程如图 9.17 所示。

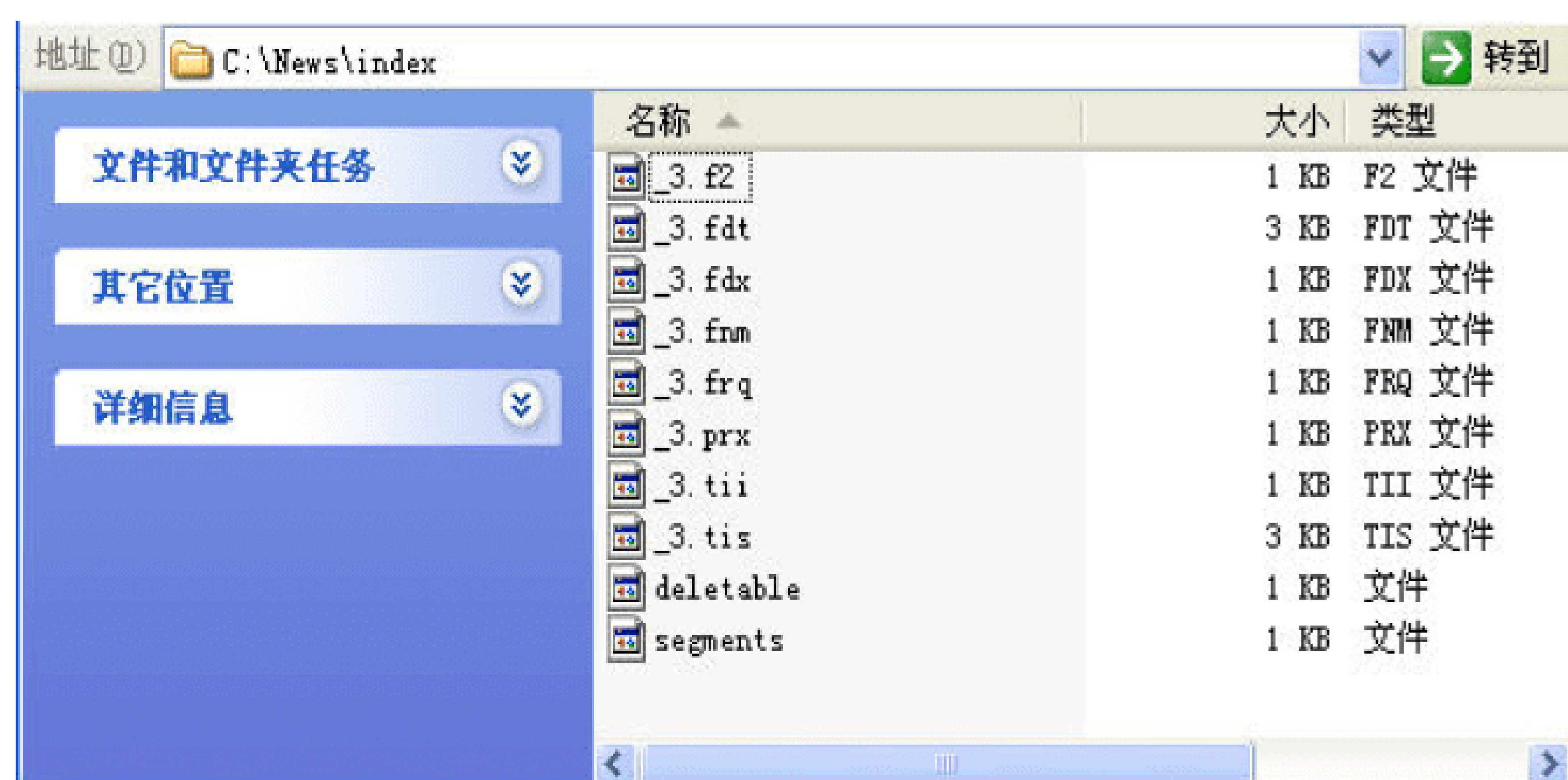


图 9.16　索引文件

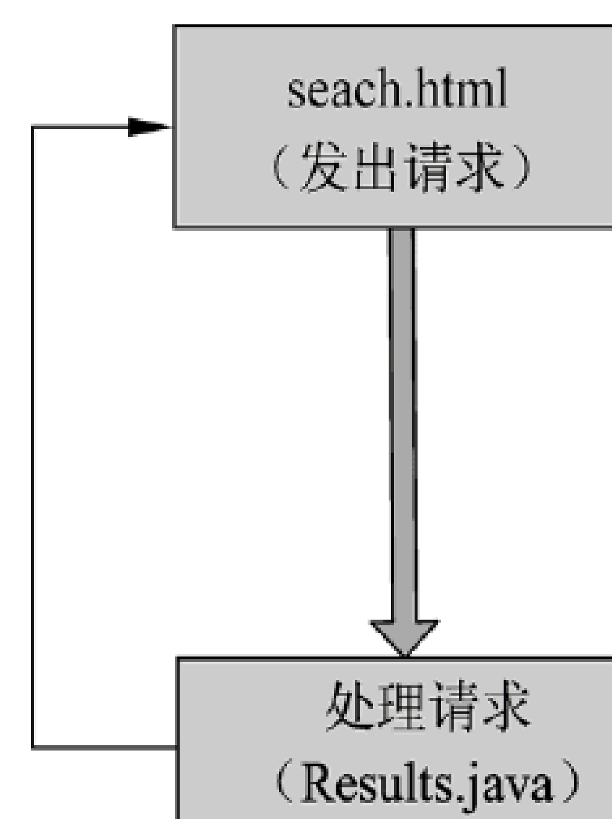


图 9.17　项目流程

Results.java 作为服务器端的 Servlet 类，用来实现查找功能，具体内容如代码 9.8 所示。

代码 9.8　查找功能：Results.java

```
...
public class Results
    extends HttpServlet {
  //定义编码格式常量
  private static final String CONTENT_TYPE = "text/html; charset=GBK";
  public void init() throws ServletException {          //初始化方法
  }
  //编写 doGet()方法
  public void doGet(HttpServletRequest request, HttpServletResponse
  response) throws
      ServletException, IOException {
    String QC = request.getParameter("QueryContent");   //获取请求的值
    if (QC == null) {                                    //判断 QC 的值
      QC = "";
    }
    else {
      QC = input(QC);
    }
    response.setContentType(CONTENT_TYPE);               //设置编码格式
    PrintWriter out = response.getWriter();              //定义输出流
    try {
      Search(QC, out);                                   //调用查找方法
    }
    catch (Exception ex) {
      System.out.println(ex.getMessage());
    }
  }
  //编写查找方法
  public void Search(String qc, PrintWriter out) throws Exception {
    IndexSearcher  searcher = new IndexSearcher("c:\\news\\index");
                                                         //从索引目录创建索引
    Analyzer analyzer = new ChineseAnalyzer();           //创建标准分析器
    String line = qc;                                    // 查询条件
    Query query = QueryParser.parse(line, "title", analyzer);
                                                         //创建抽象类
    //创建输出内容
...
    out.println("<center>" +
                "<form action='/NewsServer/results' method='get'>" +
                "<font face='华文中宋' color='#3399FF'>新闻搜索引擎</font>:" +
                "<input type='text' name='QueryContent' size='20'>" +
                "<input type='submit' name='submit' value='开始搜索'>" +
                "</form></center>"
                );
    out.println("<p>搜索关键字：<font color=red>" + query.toString
    ("title") +"</font></p>");
    Hits hits =  searcher.search(query);
    out.println(" 总共找到<font color=red>" + hits.length() + "</font>条新
    闻<br>");
    final int HITS_PER_PAGE = 10;                        //定义常量
    //遍历循环
    for (int start = 0; start < hits.length(); start += HITS_PER_PAGE) {
      int end = Math.min(hits.length(), start + HITS_PER_PAGE);
      for (int i = start; i < end; i++) {
        Document doc = hits.doc(i);
```

```
        String url = doc.get("url");
        if (url != null) {
          out.println( (i + 1) + " <a href='" + url + "'>" +
                replace(doc.get("title"), qc) +
                "</a><br>");
        }
        else {
          System.out.println("没有找到！");
        }
      }
    }
    out.println("</body></html>");
     searcher.close();
  };
  public String input(String str) {                  //解决中文问题
    String temp = null;
    if (str != null) {
      try {
        temp = new String(str.getBytes("ISO8869_1"));
      }
      catch (Exception e) {
      }
    }
    return temp;
  }
  public String replace(String title, String keyword) {
                                                     //编写 replace()方法
    return title.replaceAll(keyword, "<font color='red'>" + keyword +
    "</font>");
  };
  public void destroy() {                            //清除资源
  }
};
```

接着在 web.xml 文件中配置 Results 类，具体内容如下：

```
<!--配置 results 类路径-->
<servlet>
  <servlet-name>results</servlet-name>
  <servlet-class> Results</servlet-class>
</servlet>
<!--配置 results 映射路径-->
<servlet-mapping>
  <servlet-name>results</servlet-name>
  <url-pattern>/results</url-pattern>
</servlet-mapping>
```

【代码解析】

在上述代码中，最重要的方法为 Search()方法，在该方法中通过调用 Lucene 全文搜索的开发包，实现搜索引擎中的查找功能。

seach.html 页面为新闻搜索引擎的唯一页面，该页面的具体内容如代码 9.9 所示。

代码 9.9　新闻搜索引擎页面：seach.html

```
...
<body bgcolor="#ffffff">
<center>
 <h1><font face="华文中宋" color="#3399FF">新闻搜索引擎</font></h1>
```

```
  <form action="results" method="get">                  <!--表单-->
    <!--输入框 -->
    请输入关键字: <input type="text" name="QueryContent" size="20">
    <!--提交按钮 -->
   <input type="submit" name="submit" value="开始搜索">
  </form>
</center>
</body>
...
```

【代码解析】

在上述代码中，表单发出的请求由名为 Results 的 Servlet 程序处理。单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/newslucene/seach.html 打开首页（如图 9.18 所示）。在该页面中输入关键字“体育”，单击“开始搜索”按钮就可以出现搜索结果页面，如图 9.19 所示。



图 9.18　首页

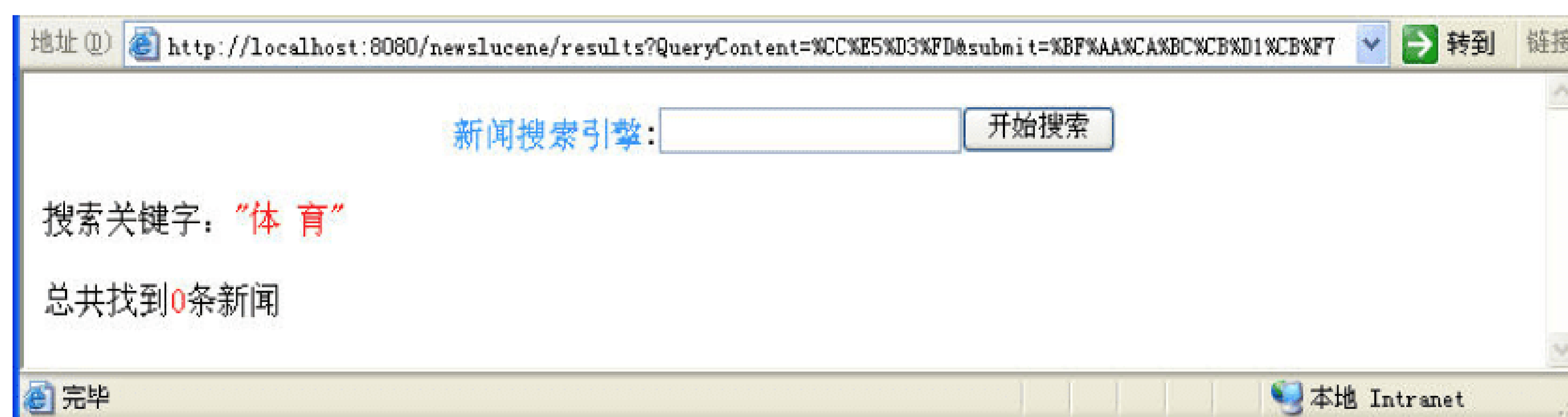


图 9.19　搜索结果

9.6　小　　结

本章主要介绍搜索引擎模块，该模块基于 JSP+Servlet 解决方案，调用 Lucene 组件和 Bot 组件等相关搜索引擎组件。

为了讲清楚搜索引擎，首先介绍了 Lucene 组件和 Bot 组件等相关搜索引擎组件，最后实现一个简单的新闻搜索引擎实例。在具体介绍相关搜索组件时，不仅介绍了搜索引擎组件的相关概念、理论、下载和配置，而且还详细介绍如何使用这些组件。在具体实现新闻搜索引擎时，首先创建所扫描网页的索引文件，然后通过调用 Lucene 组件实现新闻搜索功能。

第 10 章　在线网上支付（JSP+Servlet+JavaBean）

在线网上支付模块是针对电子商务企业类网上付费的需求而出现的，具体的电子商务体系需要具备 3 个过程：网上下单、指定配送方式和网上支付，其中最重要的过程就是网上支付模块。

本章将通过 JSP+Servlet+JavaBean 框架技术来介绍如何实现在线网上支付模块，该模块主要包含两个功能：发送信息到支付公司网关，获取支付公司网关返回信息。

10.1　在线网上支付原理

对于传统的支付方式（邮局汇款、银行电汇和信用证支付）来说，拥有的支付工具有支票、本票、汇票和信用卡等，而在线网上支付方式的工具一般是信用卡支付、电子支票。

10.1.1　在线网上支付结构框架分析

对于一个大型网上系统来说，实现一个可用的在线网上支付功能要考虑的情况十分复杂，例如采用什么方案来接入银行？如果采用通过中间公司间接接入方式时，选择哪个公司？哪个公司的安全性、费用和速度比较合理等。该章采用间接接入的方式实现在线网上支付功能，该中间公司为易宝。本系统的结构框架如图 10.1 所示，其项目目录如图 10.2 所示。

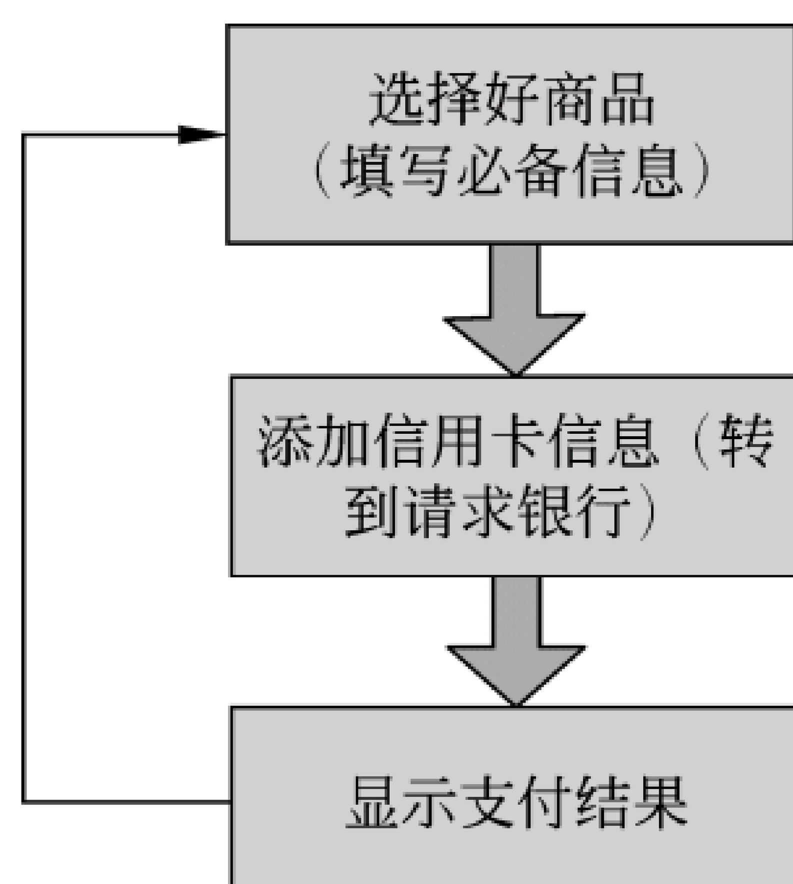


图 10.1　系统流程图

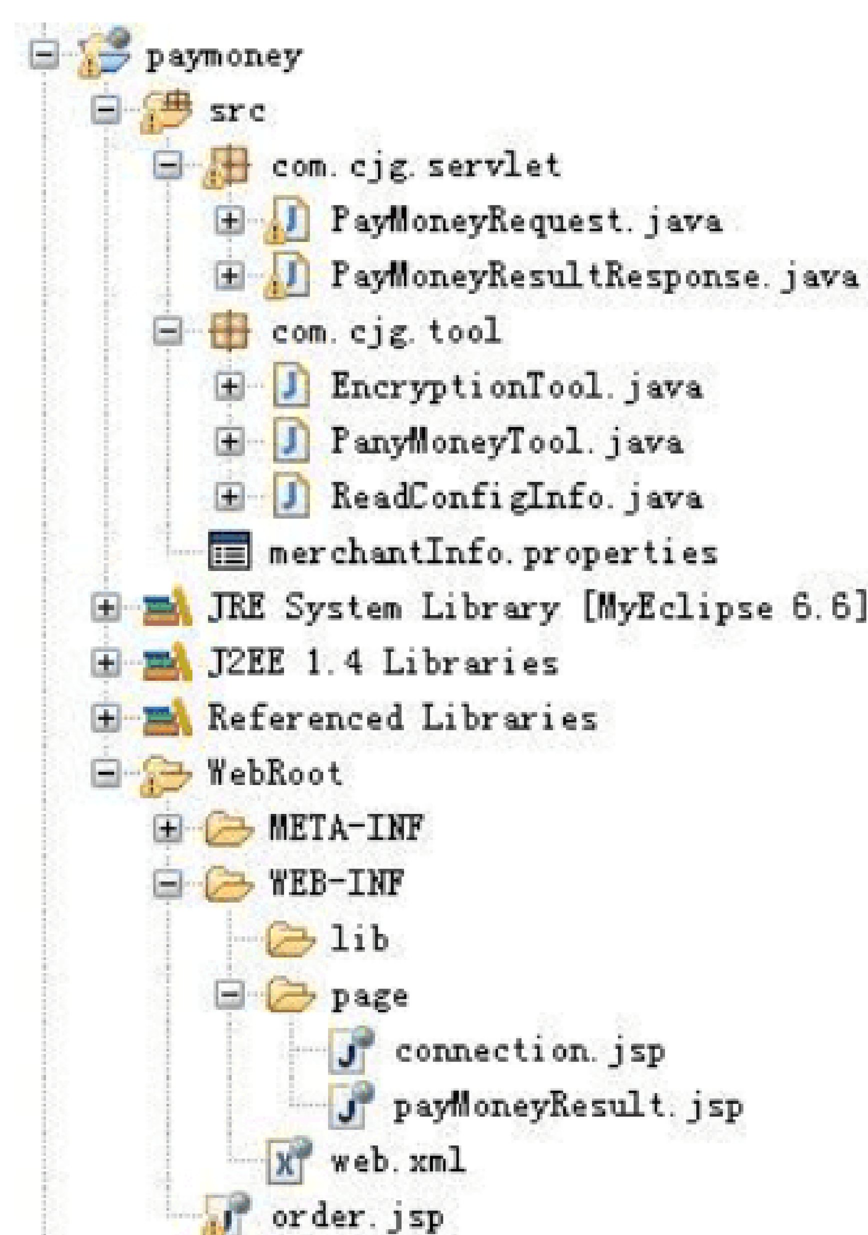


图 10.2　项目目录

10.1.2　在线网上支付功能分析

本节将以直观的方式，向读者介绍整个在线网上支付模块要实现的功能。这些功能包括选择支付银行、进行付款和返回支付结果。

1．选择支付银行

购买者通过浏览网页来选择所需要的商品后就会转到 order.jsp 页面，在该页面上会显示出具体的订单号和应付金额，这时购买者就需要在该页面中选择支付银行（工商银行）如图 10.3 所示。



图 10.3　选择支付银行

2．进行付款

当购买者选择完支付银行后，只需单击“确认支付”按钮就会转到该支付银行对应的个人网上银行页面（如图 10.4 所示）。在转到个人网上银行页面的过程中，首先会经过易宝支付网关的处理，如图 10.5 所示。

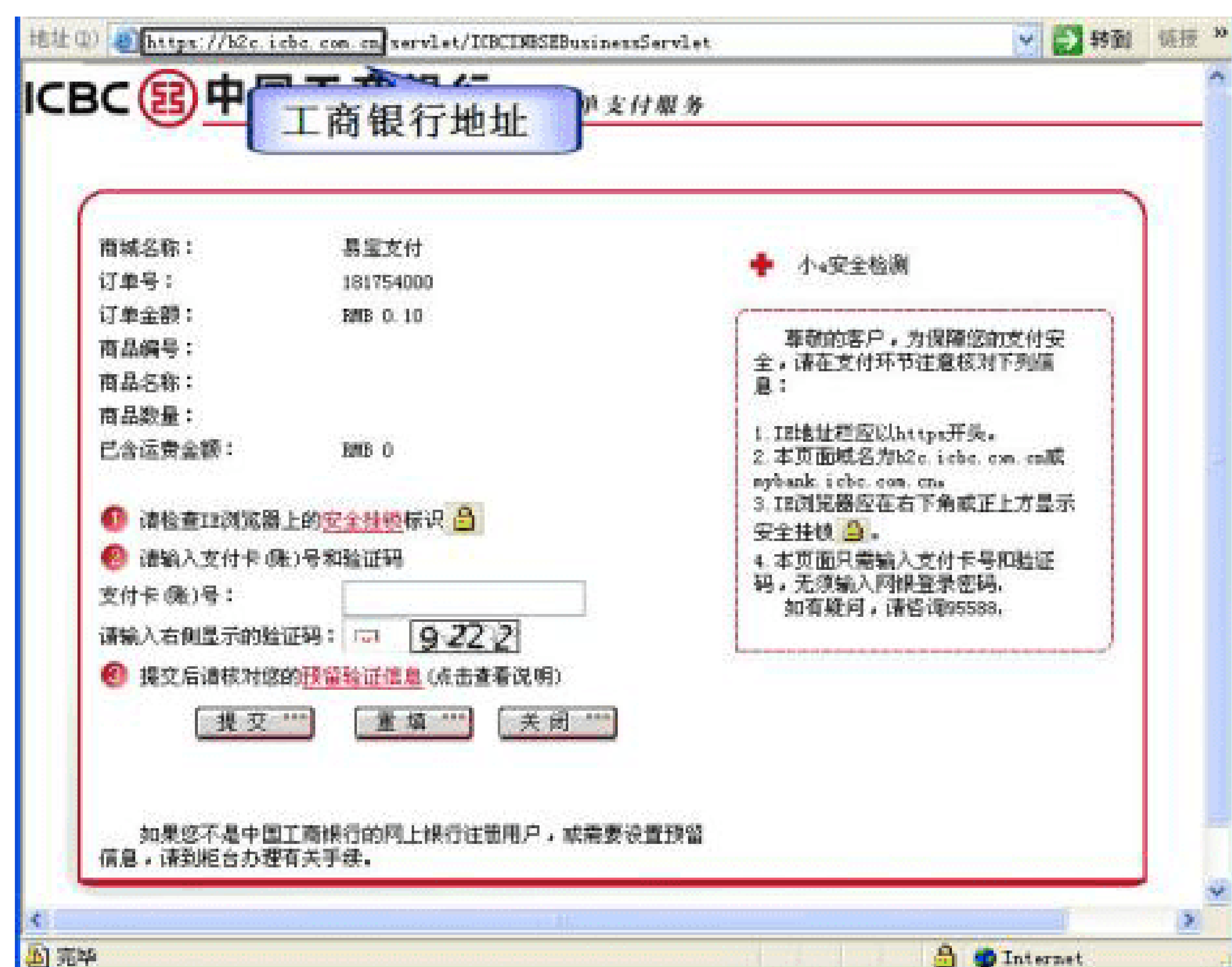


图 10.4　工商个人网上银行



图 10.5　易宝支付网关

3．返回支付结果

当购买者在图 10.4 所示工商银行的个人网上银行上填写完相应信息后，工商银行就会扣除相应的金额。当完成上述操作后就会通过易宝支付网关转到一个显示支付结果的 payMoneyResult.jsp 页面，如图 10.6 所示。

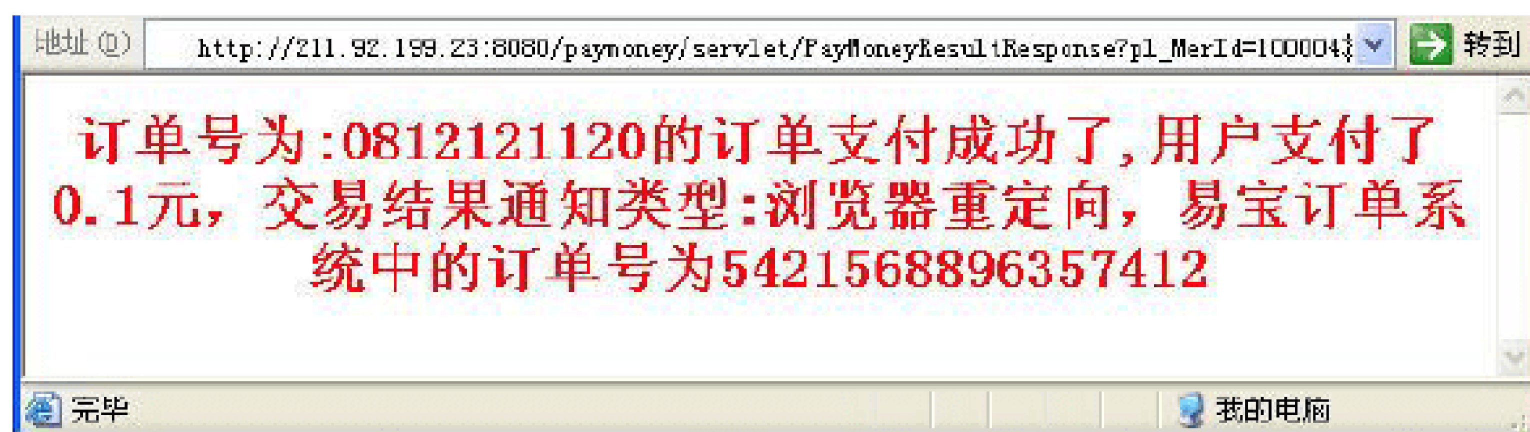


图 10.6　返回支付结果

10.1.3　在线网上支付功能的基础知识

虽然利用货币实现支付功能的方式比较麻烦，但是却非常安全、有效。对于便捷在线

网上支付系统而言，必须利用一些技术来加强交易的安全性。这些技术分别为：

- 使用数字签名和数字证书实现对各方的认证：为实现协议的安全性，对参与贸易的各方身份的有效性进行认证，通过认证机构和注册机构向参与各方发放 X.509 证书，以证实身份的合法。
- 使用加密技术对业务进行加密：可以采用对称体制和非对称体制来进行消息加密，并采用数字信封、数字签名等技术来加强数据传输的保密性，以防止未被授权的非法第三者获取消息的真正含义。
- 使用消息摘要算法以确认业务的完整性：为保护数据不被未授权者拦截、嵌入、删除、篡改、重放而完整无缺地到达接收者，可以采用数据变换技术通过对原文的变换生成消息摘要一并传送到接收者，接收者就可以通过摘要来判断所接收的消息是否完整，否则需要求发送端重发以保证其完整性。

在线网上支付模块中，最重要的部分就是如何接入银行系统。到目前为止有两种实现接入银行的方式：直接接入和间接接入。所谓直接接入就是直接与银行对接（如图 10.7 所示），所谓间接接入就是通过中间支付企业间接与银行对接（如图 10.8 所示）。两种方式的优缺点如表 10.1 所示。

表 10.1　性能比较

	优点	缺点
直接接入	交易资金比较安全，允许大的交易资金量	开发工作量大，随着银行系统的升级而需要不断更改，每年支付银行的费用比较高
间接接入	开发工作量小，不会随着银行系统的升级而不断更改，中间企业会调整。每年支付费用比较低	目前中间支付企业都是私企，资金安全性差

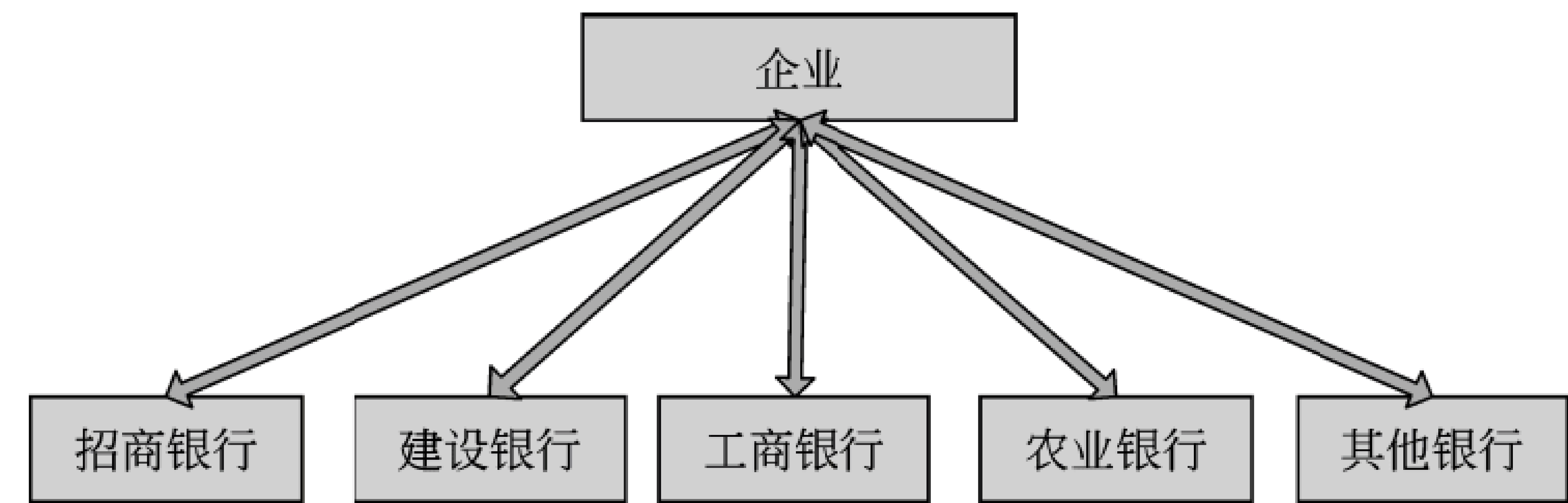


图 10.7　直接接入方式

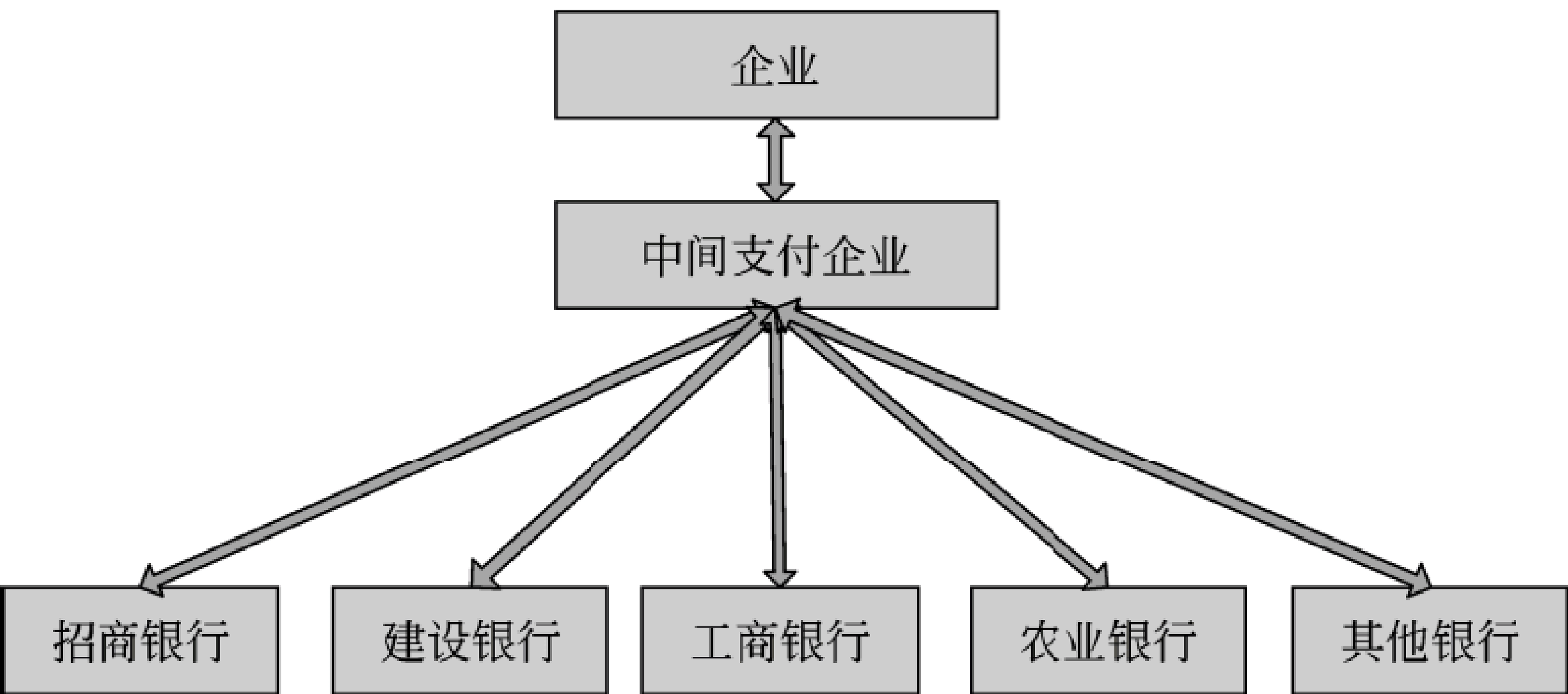


图 10.8　间接接入方式

对于中间支付企业来说，目前中国比较好的有两个，分别如下。

- 易宝支付：使用该支付接口免费，只从交易金额中扣除 1%的手续费用，例如盛大公司、e 龙网等都是使用易宝支付，该支付的网站为 http://www.yeepay.com。
- 首信易支付：使用该支付接口除了需要每年交纳一定的费用，还从交易金额中扣除 1%的手续费用。例如当当网、京东商城等都使用了首信易支付，该支付的网站为 http://www.beijing.com.cn。

如果想通过易宝支付间接接入银行，必须要了解和遵循易宝支付的接入规范。易宝支付的交换信息过程采用 MD5-hmac 加密技术，MD5-hmac 其实就是一种秘密的密钥验证算法。该种加密手段提供的数据完整性和源身份验证完全取决于密钥分配的范围，即只有发起者和接收者知道密钥。

10.2　在线网上支付功能工具类

在 paymoney 网上支付项目中，需要一些工具类来支持在线网上支付模块。分别为：实现加密算法的 EncryptionTool 类、生成 hmac 码的 PanyMoneyTool.类和读取配置信息的 ReadConfigInfo 类。

10.2.1　加密工具类

对于 MD5-hmac 加密手段来说，只有发起者和接收者才拥有密钥。发起者发起请求时，请求中传送的信息是把原信息和密钥经过 MD5 算法后而出现的 hmac 码。根据易宝支付的开发文档实现 MD5 算法，具体内容如代码 10.1 所示。

代码 10.1　加密原数据：EncryptionTool.java

```
...
public class EncryptionTool {
    private static String encodingCharset = "UTF-8";   //设置编码格式
    public static String hmacSign(String aValue, String aKey) {
                                                        //进行加密
        //创建各种数组
        byte k_ipad[] = new byte[64];
        byte k_opad[] = new byte[64];
        byte keyb[];
        byte value[];
        try {
            keyb = aKey.getBytes(encodingCharset);      //获取关于密钥的字节
            value = aValue.getBytes(encodingCharset);
                                                //获取关于所需加密字符的字节
        } catch (UnsupportedEncodingException e) {
            keyb = aKey.getBytes();
            value = aValue.getBytes();
        }
        //通过调用 Arrays.fill()方法实现加密
        Arrays.fill(k_ipad, keyb.length, 64, (byte) 54);
        Arrays.fill(k_opad, keyb.length, 64, (byte) 92);
        //通过调用 Arrays.fill()方法实现加密
        for (int i = 0; i < keyb.length; i++) {
```

```
            k_ipad[i] = (byte) (keyb[i] ^ 0x36);
            k_opad[i] = (byte) (keyb[i] ^ 0x5c);
        }
        MessageDigest md = null;              //创建MessageDigest类型变量
        try {
            md = MessageDigest.getInstance("MD5"); //为变量 md 赋值
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
        //调用 update()方法
        md.update(k_ipad);
        md.update(value);
        byte dg[] = md.digest();
        md.reset();
        //调用 update()方法
        md.update(k_opad);
        md.update(dg, 0, 16);
        dg = md.digest();
        return toHex(dg);                         //输出加密后的字符串
    }
    //把一个 byte 类型的数转换成十六进制的 ASCII 表示
    public static String toHex(byte input[]) {
...
    }
    public static String getHmac(String[] args, String key) {
...
    }
    //把字符串转换成二进制内部表示
    public static String digest(String aValue) {
...
    }
}
```

【代码解析】

- ❑ 上述代码不需要程序自己编写，从易宝支付的开发文档中就可以获取。
- ❑ 该类中的方法 hmacSign()用来生成 hmac 码，该方法有两个参数：第 1 个参数为原数据；第 2 个参数为密码。

10.2.2　生成 hmac 码工具类

在线网上支付功能最重要的两个过程就是发起支付请求和接受支付返回，当发起支付请求时，必须要把原数据和 hmac 码发送给易宝支付网关；当接收支付返回时，必须验证返回数据的身份。代码 10.2 不仅通过调用工具类 EncryptionTool 实现 hmac 码同时还实现验证 hmac 码。

代码 10.2　生成和验证 hmac 码：PanyMoneyTool.java

```
...
public class PanyMoneyTool {
    //生成 hmac 码的方法
    public static String buildHmac(String p0_Cmd,String p1_MerId,
            String p2_Order, String p3_Amt, String p4_Cur,String p5_Pid,
            String p6_Pcat,
            String p7_Pdesc,String p8_Url, String p9_SAF,String
```

```
    pa_MP,String pd_FrpId,
            String pr NeedResponse,String keyValue) {
        StringBuffer sValue = new StringBuffer();  //创建 StringBuffer 对象
        sValue.append(p0_Cmd);                     //业务类型
        sValue.append(p1_MerId);                   //商户编号
        sValue.append(p2 Order);                   //商户订单号
        sValue.append(p3_Amt);                     //支付金额
        sValue.append(p4 Cur);                     //交易币种
        sValue.append(p5_Pid);                     //商品名称
        sValue.append(p6 Pcat);                    //商品种类
        sValue.append(p7_Pdesc);                   //商品描述
        sValue.append(p8 Url);                 //商户接收支付成功数据的地址
        sValue.append(p9_SAF);                 //送货地址
        sValue.append(pa_MP);                  //商户扩展信息
        sValue.append(pd_FrpId);               //银行编码
        sValue.append(pr_NeedResponse);        //应答机制
        /调用 EncryptionTool.hmacSign()方法生成 hmac 码
        String sNewString = EncryptionTool.hmacSign(sValue.toString(), key
        Value);
        return sNewString;
    }
    //验证 hmac 码
    public static boolean verifyCallback(String hmac, String p1 MerId,
            String r0 Cmd, String r1 Code, String r2 TrxId, String r3 Amt,
            String r4 Cur, String r5 Pid, String r6 Order, String r7 Uid,
            String r8 MP, String r9 BType, String keyValue) {
        StringBuffer sValue = new StringBuffer();  //创建 StringBuffer 对象
        sValue.append(p1 MerId);                   //商户编号
        sValue.append(r0 Cmd);                     //业务类型
        sValue.append(r1 Code);                    //支付结果
        sValue.append(r2 TrxId);                   //易宝支付交易流水号
        sValue.append(r3 Amt);                     //支付金额
        sValue.append(r4 Cur);                     //交易币种
        sValue.append(r5 Pid);                     //商品名称
        sValue.append(r6 Order);                   //商户订单号
        sValue.append(r7 Uid);                     //易宝支付会员 ID
        sValue.append(r8 MP);                      //商户扩展信息
        sValue.append(r9 BType);                   //交易结果返回类型
        //生成 hmac 码
        String sNewString = EncryptionTool.hmacSign(sValue.toString(), key
        Value);
        //比较两个 hmac 码
        if (hmac.equals(sNewString)) {
            return true;
        }
        return false;
    }
}
```

【代码解析】

- 生成 hmac 码的 buildHmac()方法中，利用 append()方法连接原数据创建字符串时，原数据的顺序一定要按照易宝支付的规范顺序来生成，否则生成的 hmac 码是不一样的。
- 验证 hmac 码的 verifyCallback()，只是在接受支付返回时使用。在该方法中首先接受支付返回各个参数的值，然后调的 buildHmac()方法生成 hmac 码，最后比较新

生成的 hmac 码和返回的 hmac 码。当两者相同时，则说明身份合法；否则说明身份非法。

10.2.3　读取配置信息工具类

在该项目中存在一个名为 merchantInfo.properties 的文件，在该文件中存储着一些重要信息。为了读取该文件中的信息，需要创建一个读取信息的工具类。代码 10.3 为文件的内容。代码 10.4 实现了读取文件内容的功能。

代码 10.3　重要信息：merchantInfo.properties

```
p1_MerId=10000326625                                #商家编号 ID
#密钥
keyValue=0acqgug6x57m0wrsiod6clpn1ezh47r2ot5h1zkq5dztiic8y5xkm5g0p0ek
#接受支付后的地址
merchantCallbackURL=http\://211.92.199.23\:8080/paymoney/servlet/PayMon
eyResultResponse
```

代码 10.4　读取文件：ReadConfigInfo.java

```
...
public class ReadConfigInfo {
    private static Properties cache = new Properties();
                                        //创建一个 Properties 类型对象
    static{
        try {
    cache.load(ReadConfigInfo.class.getClassLoader().getResourceAsStream
("merchantInfo.properties"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static String getValue(String key){      //获取指定 key 的值
        return cache.getProperty(key);
    }
}
```

【代码解析】

- ReadConfigInfo.class.getClassLoader().getResourceAsStream("merchantInfo.properties")中 getResourceAsStream()方法用来读取属性文件。getClassLoader()方法获取类对象的装载器，该装载器负责将字符流读入内存。如果想调用 getClassLoader()方法必须通过当前类对象即 ReadConfigInfo。
- 对于类 Properties 来说，该类的 load()方法负责读取输入流中的所有属性列表，该类的 getProperty()方法会获取指定键值的属性值。

10.3　发出支付请求过程

本章通过 JSP+Servlet+JavaBean 框架技术来实现在线网上支付模块中的发起支付请求

过程，发起支付请求过程如图 10.9 所示，它包含两个 JSP 页面和一个 Servlet 程序：order.jsp、connection.jsp 和 PayMoney-Request.java。

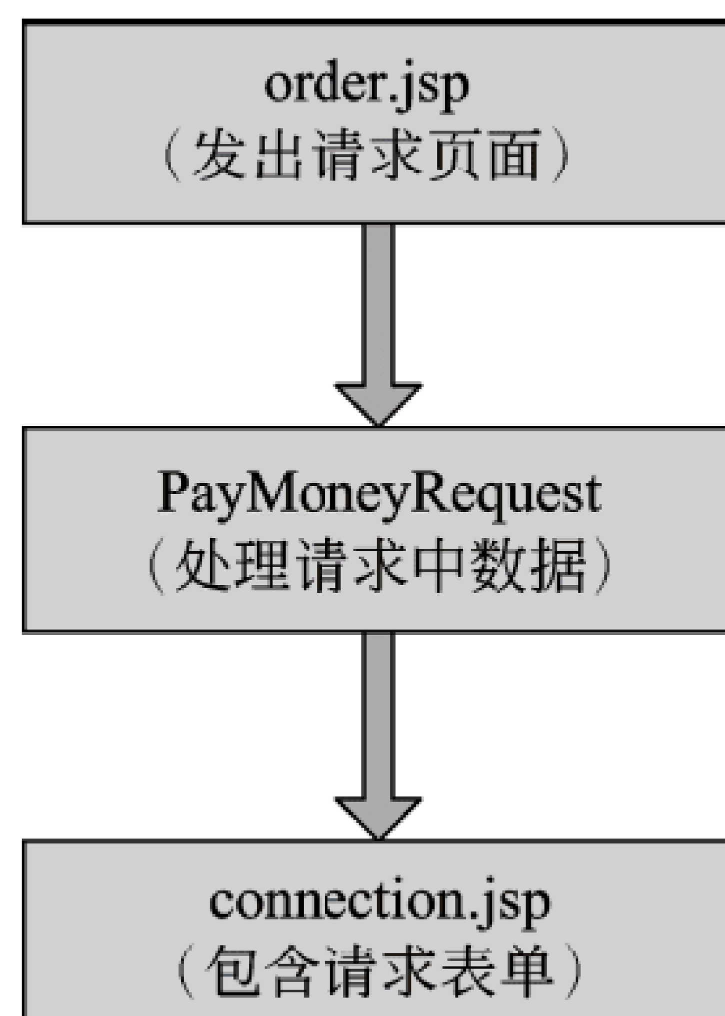


图 10.9　发起请求流程

10.3.1　发出支付请求

order.jsp 页面用来发起支付请求，当购买者选择好商品后，就会跳转到该页面。在该页面会显示出选择好的商品信息、应付的金额和有关银行的信息。购买者选择特定银行后，单击“确认支付”按钮后就会发起支付请求。代码 10.5 为发起请求的页面。

代码 10.5　订单号页面：order.jsp

```
<!--设置编码方式为 GBK-->
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
...
    <body>
                    <!--表单 -->
                    <form
    action="${pageContext.request.contextPath}/servlet/PayMoneyRequest"
                    method="post" name="paymentform">
                    <!--设置订单号和应付金额文本框-->
                    订单号：<INPUT TYPE="text" NAME="orderid">
                    应付金额：￥<INPUT TYPE="text" NAME="amount" size="6">元
                    <!--设置银行信息-->
                    <span class="STYLE3">请您选择在线支付银行</span>
                    <INPUT TYPE="radio" NAME="pd FrpId" value="CMBCHINA-
                    NET">
                    招商银行<INPUT TYPE="radio" NAME="pd_FrpId" value=
                    "ICBC-NET">
                    工商银<INPUT TYPE="radio" NAME="pd FrpId" value=
                    "ECITIC-NET">
                    中信银行
                    ...
                    <!--设置提交按钮-->
                    <input type="submit" value=" 确认支付 " />
                        </table>
                    </form>
...
    </body>
...
```

【代码解析】

- 理论上订单号和应付金额应该从数据库中获取，为了方便编写，在该项目中这两个数据需要浏览者来填写。对于易宝支付来说订单号一般为 9 位数，金额可以随便为任何值。
- 该链接参数分为两个部分，“？”前的内容为所要转移到地址，而“？”后面则为传递的参数，这些参数必须用“&”连接。

注意： 在实际编写网络购物车时，这些传递的参数值都应该是动态地从数据库中获取。

10.3.2　处理请求数据

order.jsp 页面发出的支付请求中只包含着易宝支付规范中规定的原数据，而根据易宝支付的开发文档可以知道，除了原数据外还需要加密后的 hmac 码，所以需要编写一个实现生成 hmac 码的服务器端类，代码 10.6 实现了 hmac 码的生成。

代码 10.6　生成 hmac 码：PanyMoneyTool.java

```
...
public class PayMoneyRequest extends HttpServlet {
    //编写 doGet()方法
    public  void  doGet(HttpServletRequest  request,  HttpServletResponse
    response)
            throws ServletException, IOException {
        this.doPost(request, response);
    }
    //编写 doPost()方法
    public  void  doPost(HttpServletRequest  request,  HttpServletResponse
    response)
            throws ServletException, IOException {
        request.setCharacterEncoding("GBK");                //设置编码格式
        String orderid = request.getParameter("orderid");  //订单号
        String amount = request.getParameter("amount");    //支付金额
        String pd_FrpId = request.getParameter("pd_FrpId");//选择的支付银行
        String p1_MerId = ReadConfigInfo.getValue("p1_MerId");//商家编号 ID
        String keyValue = ReadConfigInfo.getValue("keyValue");//密钥
        //接受支付返回的地址
        String merchantCallbackURL = ReadConfigInfo.getValue("merchant
        CallbackURL");
        String messageType = "Buy";         //请求命令，在线支付固定为 Buy
        String currency = "CNY";            //货币单位
        String productDesc = "";            //商品描述
        String productCat = "";             //商品种类
        String productId = "";              //商品 ID
        String addressFlag = "0";           //需要填写送货信息 0:不需要 1:需要
        String sMctProperties = "";         //商家扩展信息
        String pr_NeedResponse = "0";       //应答机制
        //调用 PanyMoneyTool.buildHmac()方法生成 hmac 码
        String md5hmac = PanyMoneyTool.buildHmac(messageType, p1_MerId,
        orderid, amount, currency,
                productId, productCat, productDesc, merchantCallbackURL,
                addressFlag, sMctProperties,
                pd_FrpId, pr_NeedResponse, keyValue);
        //设置各种属性的值
        request.setAttribute("messageType", messageType);
        request.setAttribute("merchantID", p1_MerId);
        request.setAttribute("orderId", orderid);
        request.setAttribute("amount", amount);
        request.setAttribute("currency", currency);
        request.setAttribute("productId", productId);
        request.setAttribute("productCat", productCat);
        request.setAttribute("productDesc", productDesc);
```

```
        request.setAttribute("merchantCallbackURL", merchantCallback
        URL);
        request.setAttribute("addressFlag", addressFlag);
        request.setAttribute("sMctProperties", sMctProperties);
        request.setAttribute("frpId", pd_FrpId);
        request.setAttribute("pr_NeedResponse", pr_NeedResponse);
        request.setAttribute("hmac", md5hmac);
        //把请求转发到特定

    request.getRequestDispatcher("/WEB-INF/page/connection.jsp").forward
    (request, response);
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 PayMoneyRequest 类路径-->
<servlet>
  <servlet-name>PayMoneyRequest</servlet-name>
  <servlet-class>com.cjg.servlet.PayMoneyRequest</servlet-class>
</servlet>
  <!--配置 PayMoneyRequest 映射路径-->
<servlet-mapping>
  <servlet-name>PayMoneyRequest</servlet-name>
  <url-pattern>/servlet/PayMoneyRequest</url-pattern>
</servlet-mapping>
```

【代码解析】

- 在调用 PanyMoneyTool.buildHmac()方法生成 hmac 码时，参数的顺序一定要按照易宝支付的规范顺序来生成，否则生成的 hmac 码是不一样的。
- 对于传送过来的请求中，每一个原数据不能为 null，但可以为空字符串。这是因为当该原数据为 null 时，程序会认为其为 null 字符串。
- 根据易宝支付的开发文档可以知道，当向易宝支付发起一个支付请求时，必须使用表单以 past 方式向易宝支付网关发起一个支付请求。所以首先必须把原数据和 hmac 码存储起来，然后通过 request.getRequestDispatcher()方法把请求转发到指定的表单页面 connection.jsp。

查阅易宝支付的开发文档，可以得到向易宝支付网关（https://www.yeepay.com/app-merchant-proxy/node）发起请求的 form 表单的编写方法，代码 10.7 实现了发起请求的表单。

代码 10.7　发起请求的表单：connection.jsp

```
<!--设置编码方式-->
<%@ page language="java" pageEncoding="GBK"%>
...
    <!--设置自动提交表单-->
    <body onload="javascript:document.forms[0].submit()">
    <!--表单-->
    <form  name="yeepay"  action="https://www.yeepay.com/app-merchant-
    proxy/node" method='post'>
        <!-- 请求命令，在线支付固定为 Buy -->
        <input type='hidden' name='p0_Cmd'  value="${messageType}">
        <input type='hidden' name='p1_MerId' value="${merchantID}">
        <!-- 商家 ID -->
        <input type="hidden" name="p2_Order" value="${orderId}"> <!-- 商
```

```
        家的交易订单号 -->
        <input type='hidden' name='p3 Amt'  value="${amount}">
        <!-- 订单金额 -->
        <input type='hidden' name='p4 Cur'  value="${currency}">
        <!-- 货币单位 -->
        <input  type='hidden'  name='p5 Pid'      value="${productId}">
            <!-- 商品 ID -->
        <input  type='hidden'  name='p6_Pcat'    value="${productCat}">
            <!-- 商品种类 -->
        <input type='hidden' name='p7 Pdesc' value="${productDesc}">
        <!-- 商品描述 -->
        <!-- 交易结果通知地址 -->
        <input type='hidden' name='p8_Url'  value="${merchantCallback
        URL}">
        <!-- 需要填写送货信息 0：不需要 1:需要 -->
        <input type='hidden' name='p9 SAF'  value="${addressFlag}">
        <!-- 商家扩展信息 -->
        <input type='hidden' name='pa MP'   value="${sMctProperties}">
        <input type='hidden' name='pd FrpId' value="${frpId}">
        <!-- 银行 ID -->
        <!-- 应答机制 为“1”：需要应答机制;为“0”：不需要应答机制 -->
        <input type="hidden" name="pr_NeedResponse"  value="0">
        <input type='hidden' name='hmac' value="${hmac}"><!-- MD5-hmac 验
        证码 -->
    </form>
  </body>
</html>
```

【代码解析】

上述代码的相关解释可以查看易宝支付的帮助文档，需要注意的是如何利用 JavaScript 语言实现表单自动提交。

注意： 由于购买者不需要访问 connection.jsp 页面的信息，所以把该页面创建到 paymoney/WEB-INF/page 文件夹中。

至此，当购买者单击“确认支付”按钮后就可以通过易宝支付网关转到相应的个人网上银行。

10.4　接受支付返回过程

在 10.3 节介绍了如何实现发起支付请求，本节将通过 JSP+Servlet+JavaBean 框架技术来实现接受支付响应，其流程如图 10.10 所示。它包含一个 JSP 页面和一个 Servlet 程序：payMoneyResult.jsp 和 PayMoneyResultResponse.java。

10.4.1　接受支付响应

当在相应的个人网上银行上填写个人信息后，经过在线支付系统的处理后会把结果返回给易宝支付网关，该网关会根据配置信息判断传递过来的支付信息是否为成功信息。即

使支付成功后，当易宝支付网关接受到支付信息后，也一定要判断数据来源的身份是否合法。代码 10.8 用来响应银行支付结果。

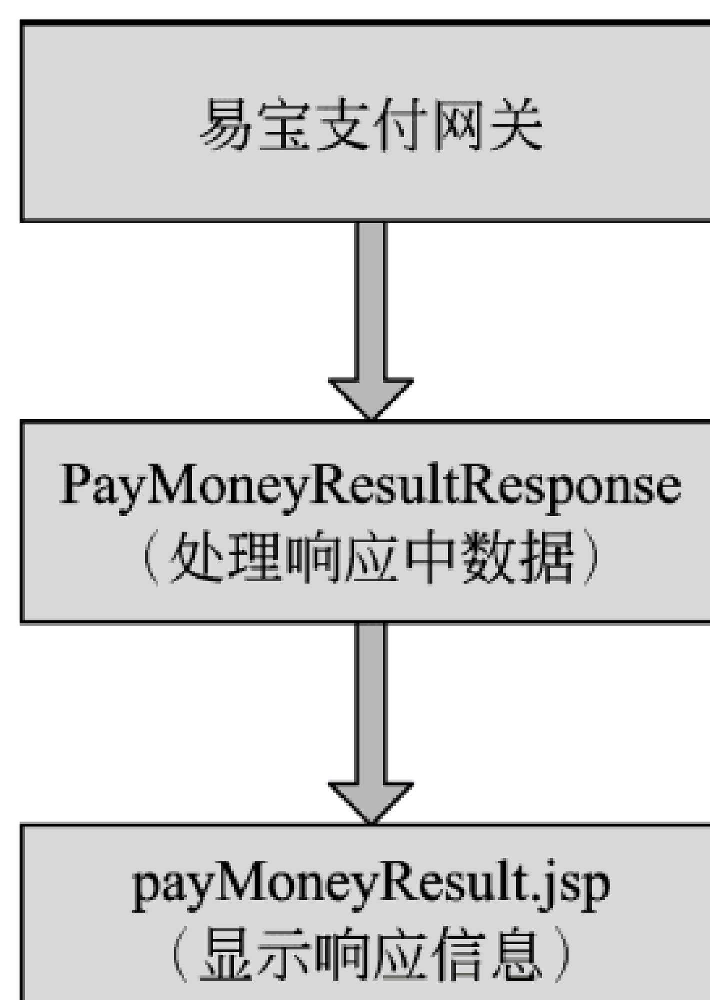


图 10.10　接受支付响应流程

代码 10.8　响应银行支付结果请求：PayMoneyRequest.java

```
...
public class PayMoneyResultResponse extends HttpServlet {
    //编辑 doGet()方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        this.doPost(request, response);
    }
    //编辑 doPos()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        request.setCharacterEncoding("GBK");
        String merchantID = ReadConfigInfo.getValue("p1 MerId");
                                                        //商家 ID
        String keyValue = ReadConfigInfo.getValue("keyValue");
                                                        //商家密钥
        String sCmd = request.getParameter("r0_Cmd");       //业务类型
        //扣款结果，该字段值为 1 时表示扣款成功.
        String sResultCode = request.getParameter("r1_Code");
        String sTrxId = request.getParameter("r2 TrxId");
                                                //YeePay 易宝交易订单号
        //扣款金额，交易结束后，YeePay 易宝交易系统将实际扣款金额返回给商户
        String amount = request.getParameter("r3 Amt");
        String currency = request.getParameter("r4 Cur");
                                                //交易币种，人民币为 CNY
        String productId = request.getParameter("r5_Pid"); //商品 ID
        String orderId = request.getParameter("r6_Order"); //商户订单号
        String userId = request.getParameter("r7 Uid");
                                                    //YeePay 易宝会员 ID
        //商户扩展信息，可以任意填写 1K 的字符串，交易返回时将原样返回
        String mp  = request.getParameter("r8 MP");
        //交易结果通知类型，1：交易成功回调（浏览器重定向）2：交易成功主动通知(服务
        //器点对点通信）
        String bType = request.getParameter("r9_BType");
```

```
        String rb_BankId  = request.getParameter("rb_BankId"); //支付银行
        String rp_PayDate = request.getParameter("rp_PayDate");
                                                    //在银行支付时的时间
        String hmac = request.getParameter("hmac");    //MD5 交易签名
        boolean result = PanyMoneyTool.verifyCallback(hmac, merchantID,
        sCmd, sResultCode, sTrxId, amount,
                currency, productId, orderId, userId, mp, bType, keyValue);
        if(result){
            if("1".equals(sResultCode)){
                //设置输出字符串
                String message = "订单号为:"+ orderId+ "的订单支付成功了";
                message += ",用户支付了"+ amount +"元";
                message +=",交易结果通知类型:";
                if("1".equals(bType)){                    //判断
                     message += "浏览器重定向";
                }else if("2".equals(bType)){
                     message += "易宝支付网关后台程序通知";
                }
                message += ",易宝订单系统中的订单号为:"+ sTrxId;
                request.setAttribute("message", message);
            }else{
                request.setAttribute("message", "用户支付失败");
            }
        }else{
            request.setAttribute("message", "数据来源不合法");
        }

    request.getRequestDispatcher("/WEB-INF/page/payMoneyResult.jsp").for
    ward(request, response);
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
    <!--配置 PayMoneyResultResponse 类路径-->
    <servlet>
        <servlet-name>PayMoneyResultResponse</servlet-name>
        <servlet-class>
            com.cjg.servlet.PayMoneyResultResponse
        </servlet-class>
    </servlet>
    <!--配置 PayMoneyResultResponse 映射路径-->
    <servlet-mapping>
        <servlet-name>PayMoneyResultResponse</servlet-name>
        <url-pattern>/servlet/PayMoneyResultResponse</url-pattern>
    </servlet-mapping>
```

【代码解析】

- 在上述代码中，首先获取易宝网关返回参数的值，然后通过 PanyMoneyTool.verifyCallback()方法检验身份的合法性。
- 当身份合法和不合法时分别设置一些属性的值，然后转向到 payMoneyResult.页面显示设置的属性值。

10.4.2　显示银行支付结果

当购买者填写完个人网上银行的相应信息后，银行系统会自动扣除相应的金额。可是

由于一些客观的原因有时银行系统会出现扣除金额不成功的情况，于是为用户显示一些重要的响应参数是非常必要的，代码 10.9 用来显示响应信息。

代码 10.9　响应支付网关结果：payMoneyResult.jsp

```
<!--设置编码方式-->
<%@ page language="java" pageEncoding="GBK"%>
...
    <body >
        <center><h3><font color="red">
        ${message }                                    <!--显示支付结果信息-->
        </font></h3>
        </center>
    </body>
...
```

注意： 由于除购买者外不需要访问 payMoneyResult.jsp 页面的信息，所以也把该页面创建到 paymoney/WEB-INF/page 文件夹中。

10.5　小　　结

本章主要介绍在线网上支付模块，该模块基于 JSP+Servlet+JavaBean 解决方案，保持了良好的 Java EE 中 MVC 分层思想。

在线支付模块在业务上主要分成两部分：发出支付请求过程和接受支付返回结果过程。在具体实现各个功能之前，首先创建被它们调用的各种工具类：实现加密工具类 EncryptionTool、生成 hmac 码工具类 PanyMoneyTool 和读取配置信息工具类 ReadConfigInfo。接着创建发出支付请求过程，该过程由两部分构成：发出支付请求和处理请求数据。最后创建处理支付返回结果过程，该过程由两部分构成：接受支付返回和显示返回结果。

第 11 章　Java Web 邮件发送系统（JSP+Servlet+JavaBean）

Java Web 邮件模块对于大型网站系统来说是一个非常重要和常见的模块，最常见功能：如果浏览者在网站上是第一次注册，为了安全保密，则会将密码以电子邮件的方式发给用户。如果用户忘记密码时，浏览者只需要在网站上填入自己的 E-mail 地址，网站系统就会自动把密码用电子邮件发送给浏览者。

本章将通过 JSP+Servlet+JavaBean 框架技术来介绍如何实现 Java Web 邮件发送系统。该系统可以实现如下功能：普通方式发送电子邮件、HTML 方式发送电子邮件、携带附件（TXT、DOC）方式发送电子邮件。

11.1　Java Web 邮件发送系统原理

Java Web 邮件发送系统的功能实际上跟 126、163、新浪和雅虎等邮件服务器的客户端相关写信功能一样，只需要用户在邮件页面上填写相应信息后，单击“发送邮件”按钮就可以自动发送该邮件到相应的用户。

11.1.1　Java Web 邮件发送结构框架分析

对于一个大型网上邮件系统来说，实现一个可用的网络发送系统要考虑的情况十分复杂，例如如何让用户在写信时尽量人性化、如何提高发送邮件内容大小等。本系统的结构框架如图 11.1 所示。

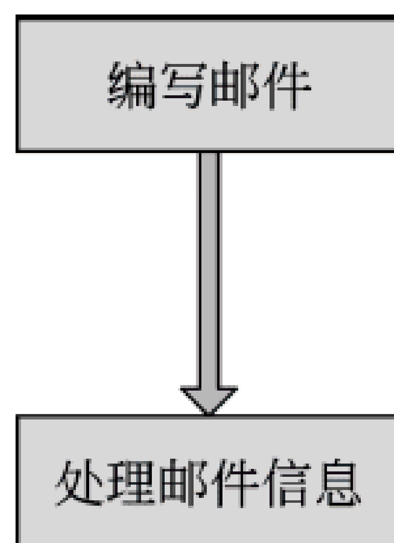


图 11.1　系统流程图

11.1.2　Java Web 邮件发送功能描述

在本节中，将以直观的方式向读者介绍各种邮件系统的功能。这些功能包括普通方式的电子邮件发送、HTML 方式电子邮件发送和携带附件电子邮件发送。

1．普通方式电子邮件的发送

浏览者首先打开 simplemail.html 页面，在该页面中输入所要发送的电子邮件的相应信息。例如：From 文本框中为 cjgong@microsof-3593ea.com、TO 文本框中为 cjgong_1@microsof-3593ea.com、Subject 文本框中为“测试简单邮件”和 Context 文本框中

为“这是简单邮件测试！！！”，如图 11.2 所示。单击“发送邮件”按钮，就会转到如图 11.3 所示成功发送后的页面。



图 11.2　输入相应内容

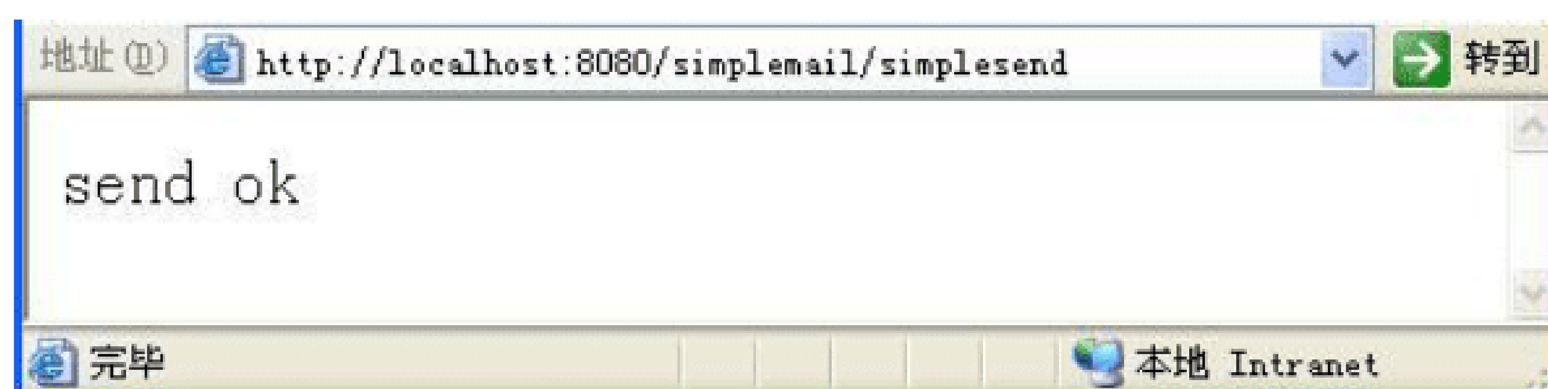


图 11.3　电子邮件发送成功

如果想查看电子邮件是否发送成功，可以首先打开邮件服务器（CmailServer），如图 11.4 所示。在该软件的界面中，可以发现账号 cjgong_1 的邮件个数为 2，而信息栏里出现发送成功的信息：

SMTP WebMail mail from cjgong@microsof-3593ea.com to cjgong_1@microsof-3593ea.com successfully。



图 11.4　邮件服务器界面

如果想查看相关邮件的内容，可以通过邮件服务器（CmailServer）的后台程序进入账户 cjgong_1 的目录中（如图 11.5 所示），在该目录中单击关于“测试简单邮件”主题的电子邮件链接后，就会进入如图 11.6 所示的显示该电子邮件的页面。



图 11.5　电子邮件目录



图 11.6　显示邮件内容

2．HTML方式电子邮件的发送

浏览者首先打开 htmlmail.html 页面，在该页面中输入所要发送的电子邮件的相应信息。例如：From 文本框中为 cjgong_1@microsof-3593ea.com、TO 文本框中为 cjgong@microsof-3593ea.com、Subject 文本框中为“测试 Html 方式邮件”和 Context 中为“<html><body>这是测试 Html 方式邮件！！！</body></html>”，最后在为 type 选择 HTML 选项，如图 11.7 所示。单击“发送邮件”按钮，就会转到如图 11.8 所示成功发送后的页面。

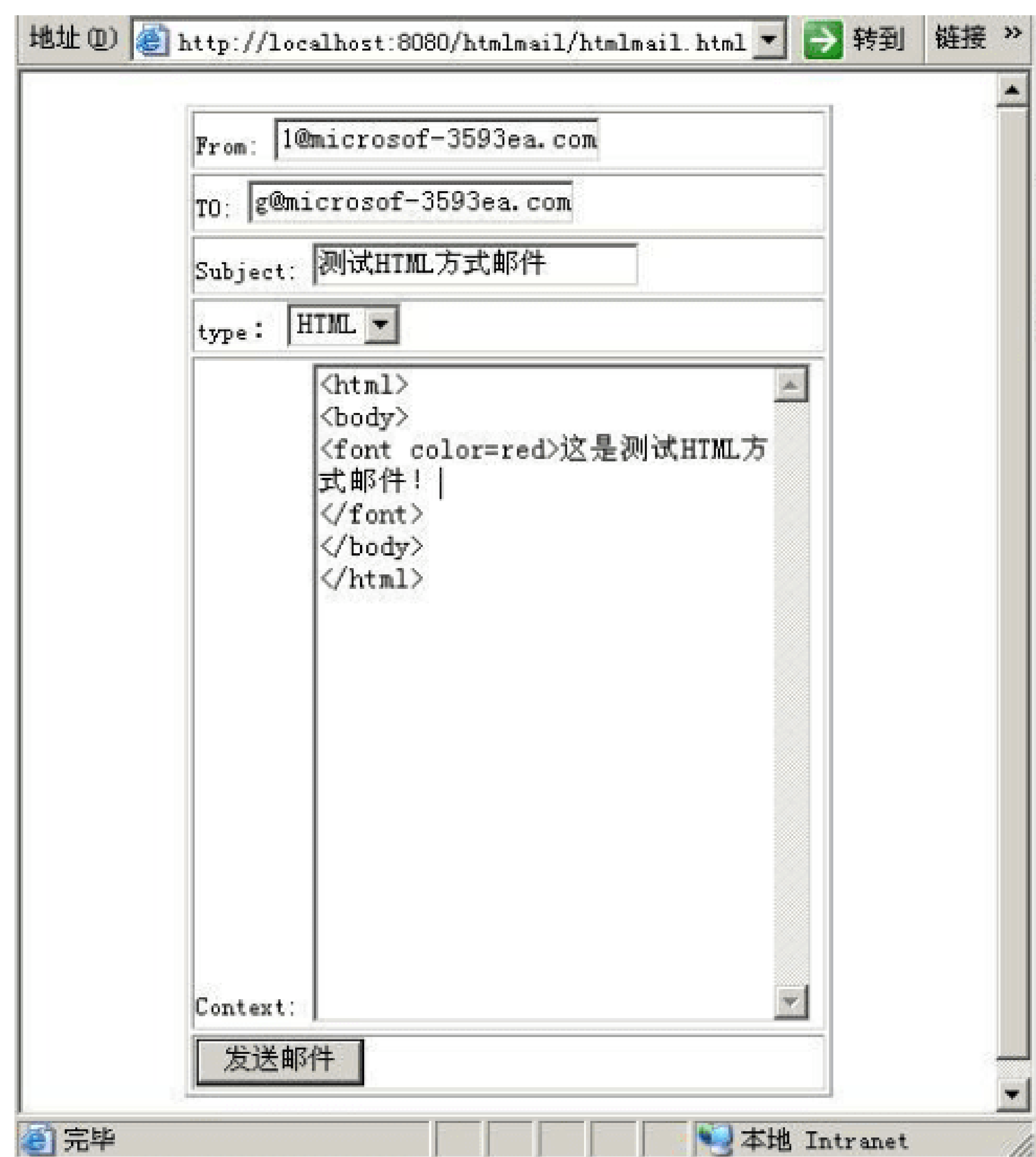


图 11.7　输入相应邮件内容

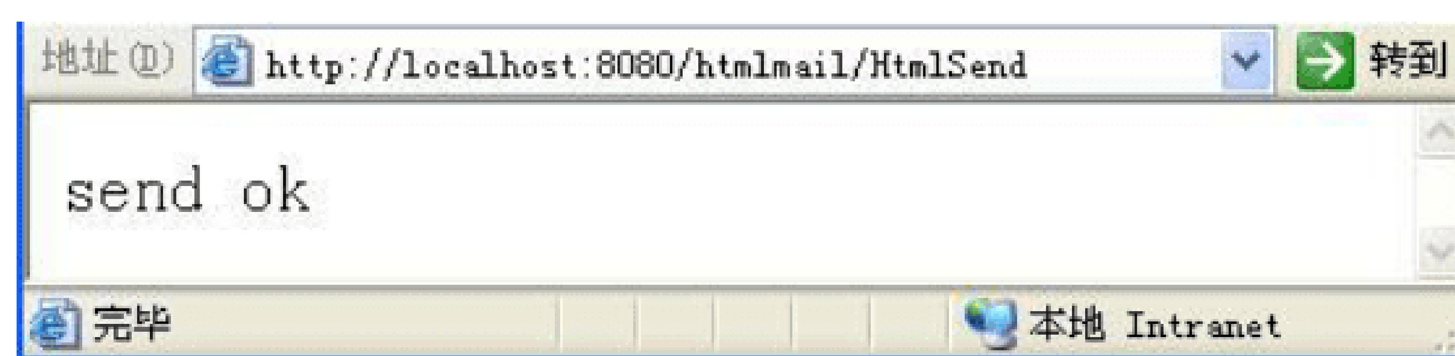


图 11.8　发送成功页面

如果想查看电子邮件是否发送成功，可以首先打开邮件服务器（CmailServer），如图 11.9 所示。在该软件的界面中，可以发现账号 cjgong 的邮件个数为 2，而信息栏里出现发送成功的信息为：

SMTP WebMail mail from cjgong_1@microsof-3593ea.com to cjgong@microsof-3593ea.com successfully。

如果想查看相关邮件的内容，可以通过邮件服务器（CmailServer）的后台程序进入账户 cjgong 的目录中，在该目录中单击关于“测试 HTML 方式邮件”主题的电子邮件链接

后，就会进入如图 11.10 所示的显示该电子邮件的页面。



图 11.9　电子邮件目录



图 11.10　显示邮件内容

如果在填写完相应的邮件内容同时为 type 选择“Text”选项，那么单击“发送邮件”按钮就会发送成功。这时候打开邮件服务器（CmailServer）查看电子邮件，如图 11.11 所示。在该软件的界面中，可以发现账号 cjgong 的邮件个数为 3，而信息栏里出现发送成功的信息为：

WebMail mail from cjgong_1@microsof-3593ea.com to cjgong@microsof-3593ea.com successfully。



图 11.11　电子邮件目录

如果想查看相关邮件的内容，可以通过邮件服务器（CmailServer）的后台程序进入账户 cjgong 的目录里，在该目录里单击关于“测试 HTML 方式邮件”主题的电子邮件链接后，就会进入如图 11.12 所示的显示该电子邮件的页面。

3．携带附件电子邮件的发送

浏览者首先打开 filemail.html 页面，在该页面中输入所要发送的电子邮件的相应信息。

例如：From 文本框中为 cjgong@microsof-3593ea.com、TO 文本框中为 cjgong_1@microsof-3593ea.com、Subject 文本框中为“携带附件”和 Context 中为“测试携带附件的邮件！！！”，最后选择携带的文件，如图 11.13 所示。单击“发送邮件”按钮，就会转到如图 11.14 所示成功发送后的页面。



图 11.12　显示邮件内容

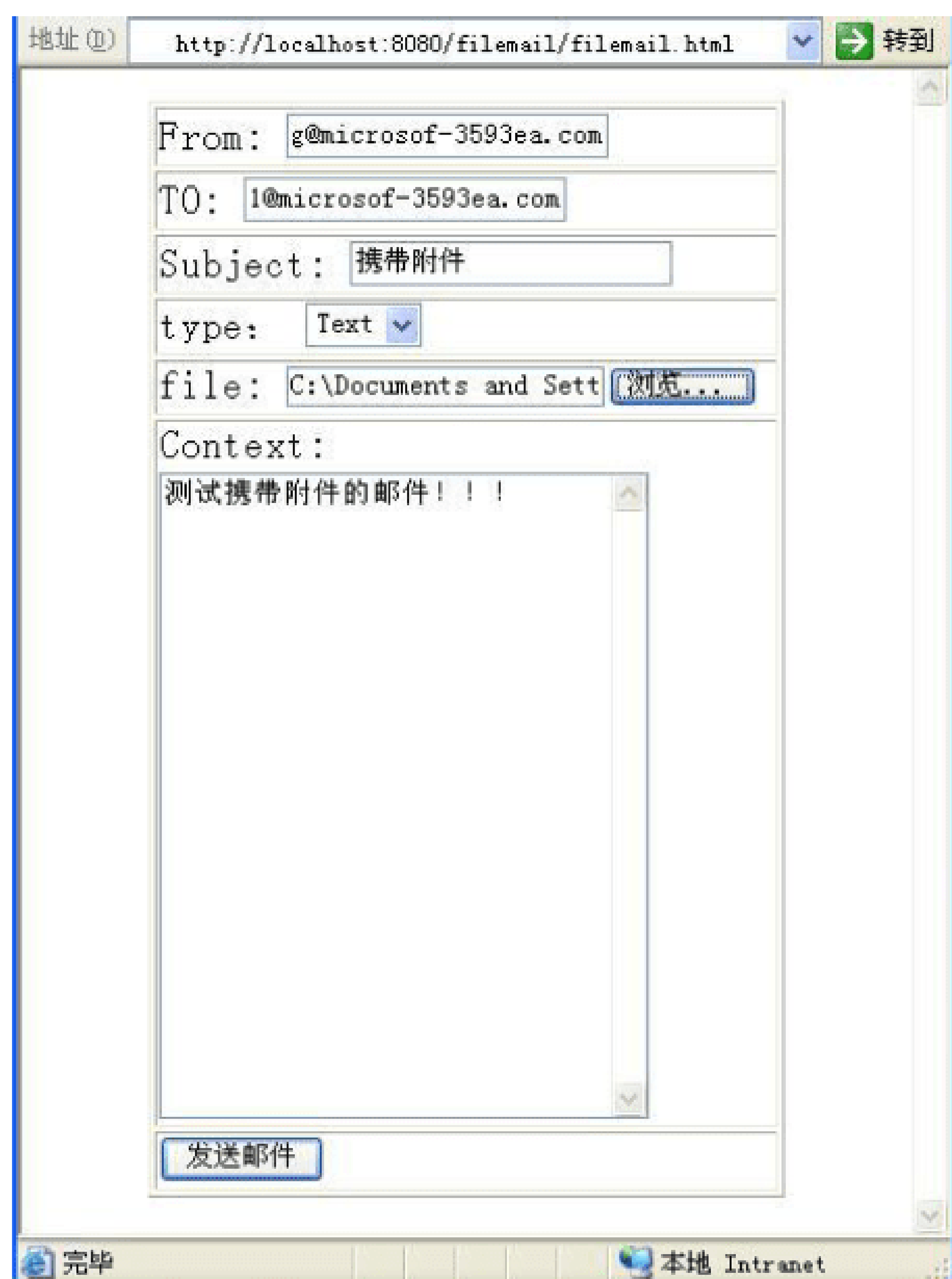


图 11.13　输入相应邮件内容

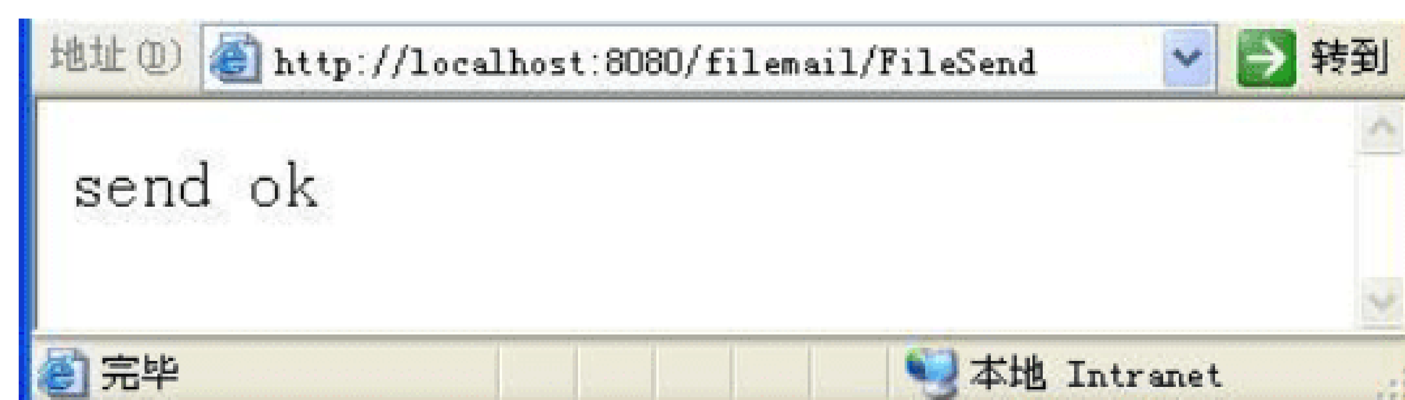


图 11.14　发送成功页面

如果想查看电子邮件是否发送成功，可以首先打开邮件服务器（CmailServer），如图 11.15 所示。在该软件的界面中，可以发现账号 cjgong_1 的邮件个数为 3，而信息栏里出现发送成功的信息为：

SMTP WebMail mail from cjgong@microsof-3593ea.com to cjgong_1@microsof-3593ea.com successfully。

如果想查看相关邮件的内容，可以通过邮件服务器（CmailServer）的后台程序进入账户 cjgong_1 的目录里（如图 11.16 所示），在该目录里单击关于“携带附件”主题的电子

邮件链接后，就会进入如图 11.17 所示的显示该电子邮件的页面。在显示邮件内容的页面中，单击附件后面的超级链接就会打开“文件下载”对话框（如图 11.18 所示），在该对话框中单击“保存”按钮就可以下载该附件。



图 11.15　邮件服务器界面



图 11.16　电子邮件列表



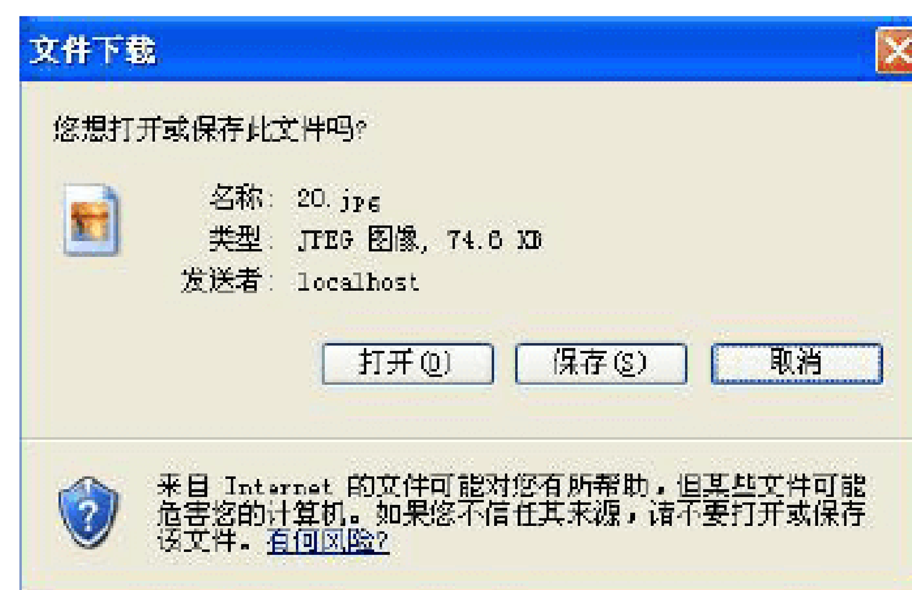


图 11.17　信件的详细内容　　　　图 11.18　文件下载

11.2　下载邮件相关 jar 包

JavaMail 组件是 Sun 公司发布的用来处理 E-mail 的 API，利用其可以方便地执行一些常用的邮件传输功能。虽然 JavaMail 是 Sun 的 API 之一，但它目前还没有被加在标准的 Java 开发工具包中（Java Development Kit），这就意味着在使用前必须下载 JavaMail 文件。除此以外，还需要 Sun 公司的 JavaBeans Activation Framework（JAF）组件，利用其可以使对邮件的操作变得更加简单易用。

11.2.1　下载 JavaMail 组件

JavaMail 组件是 Sun 项目组开发的一个功能非常强大的处理 E-mail 组件，该组件主要用来实现邮件发送功能，目前最新的版本为 JavaMail1.4.2，具体下载步骤如下。

（1）首先访问 Sun 的官方网站（http://www.java.sun.com/），如图 11.19 所示。在该页面中单击 Downloads 按钮就可以进入关于 Sun 的下载产品页面。



图 11.19　Sun 官方首页

（2）在关于 Sun 的下载产品页面中（如图 11.20 所示），单击 Java EE 下面的 JavaMail 产品就会进入关于 JavaMail 产品的页面。

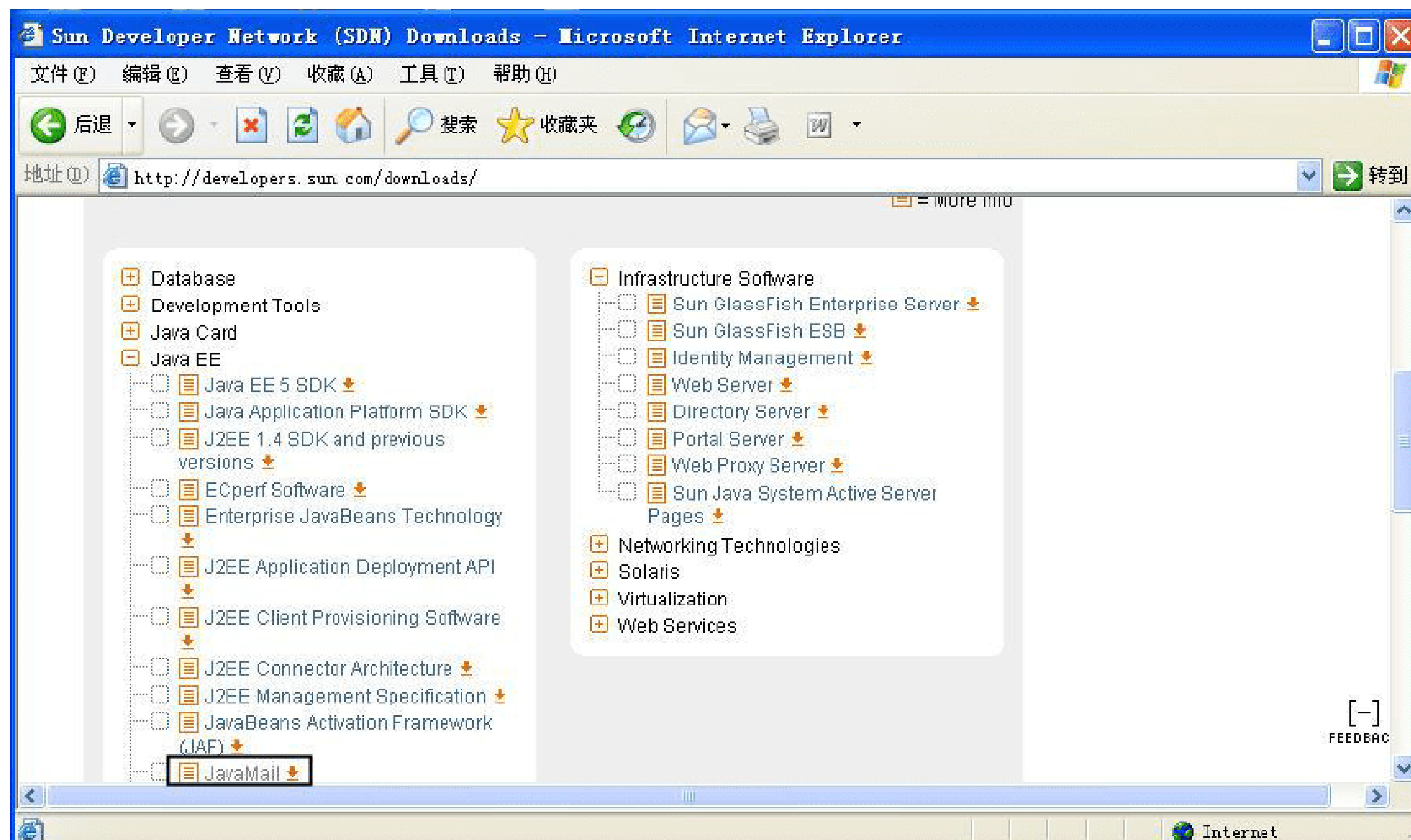


图 11.20　选择 JavaMail 产品

（3）关于 JavaMail 产品的页面中（如图 11.21 所示），单击 Download 按钮就会进入关于下载 JavaMail 的页面。

（4）在关于下载 JavaMail 的页面（如图 11.22 所示）中，对于 Platform 选择框则选中

Generic 选项，而 Language 选择框则选中 English 选项。然后单击 Continue 按钮就会进入下载 JavaMail 的真正页面。

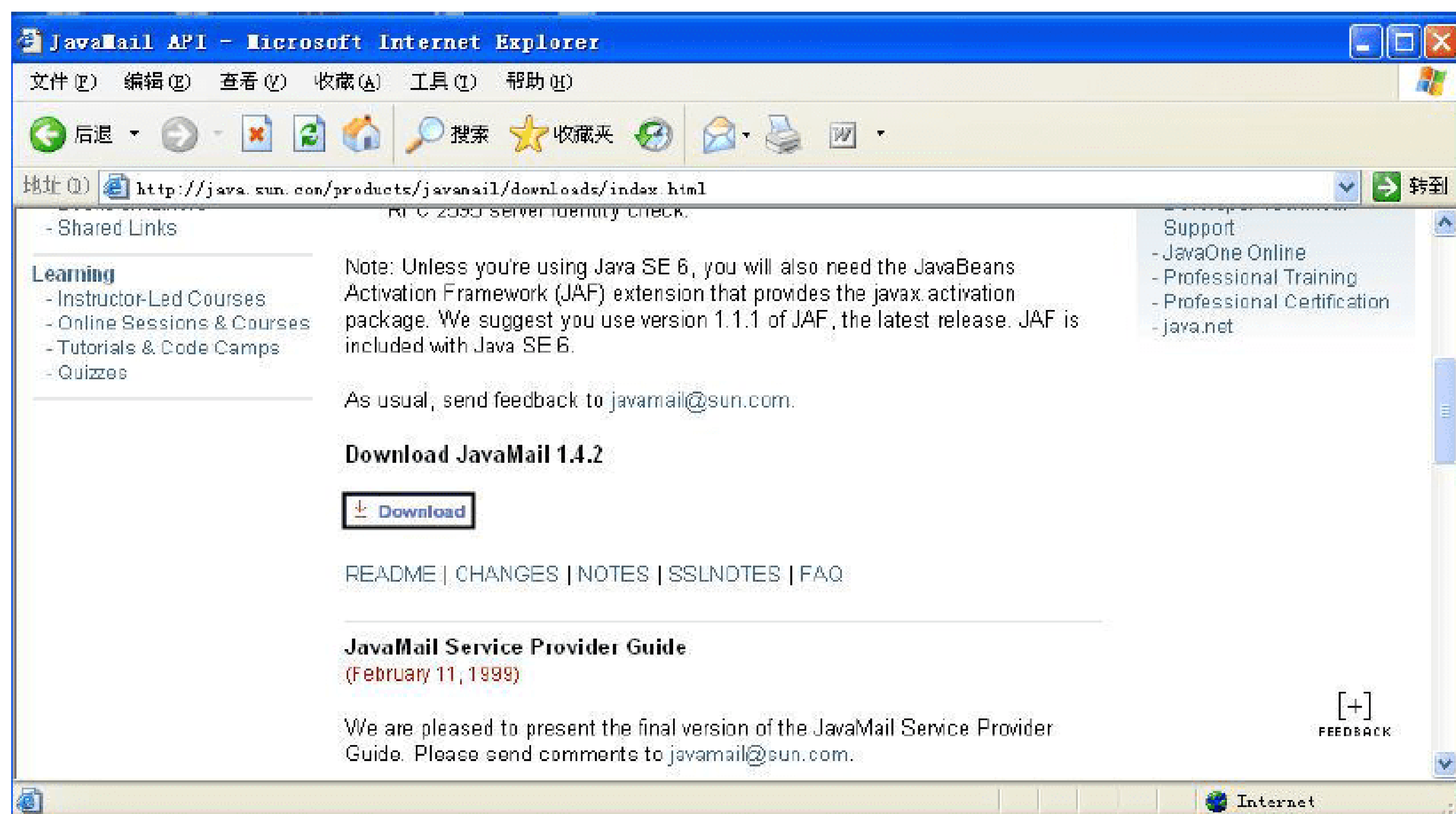


图 11.21　关于 JavaMail 产品页面

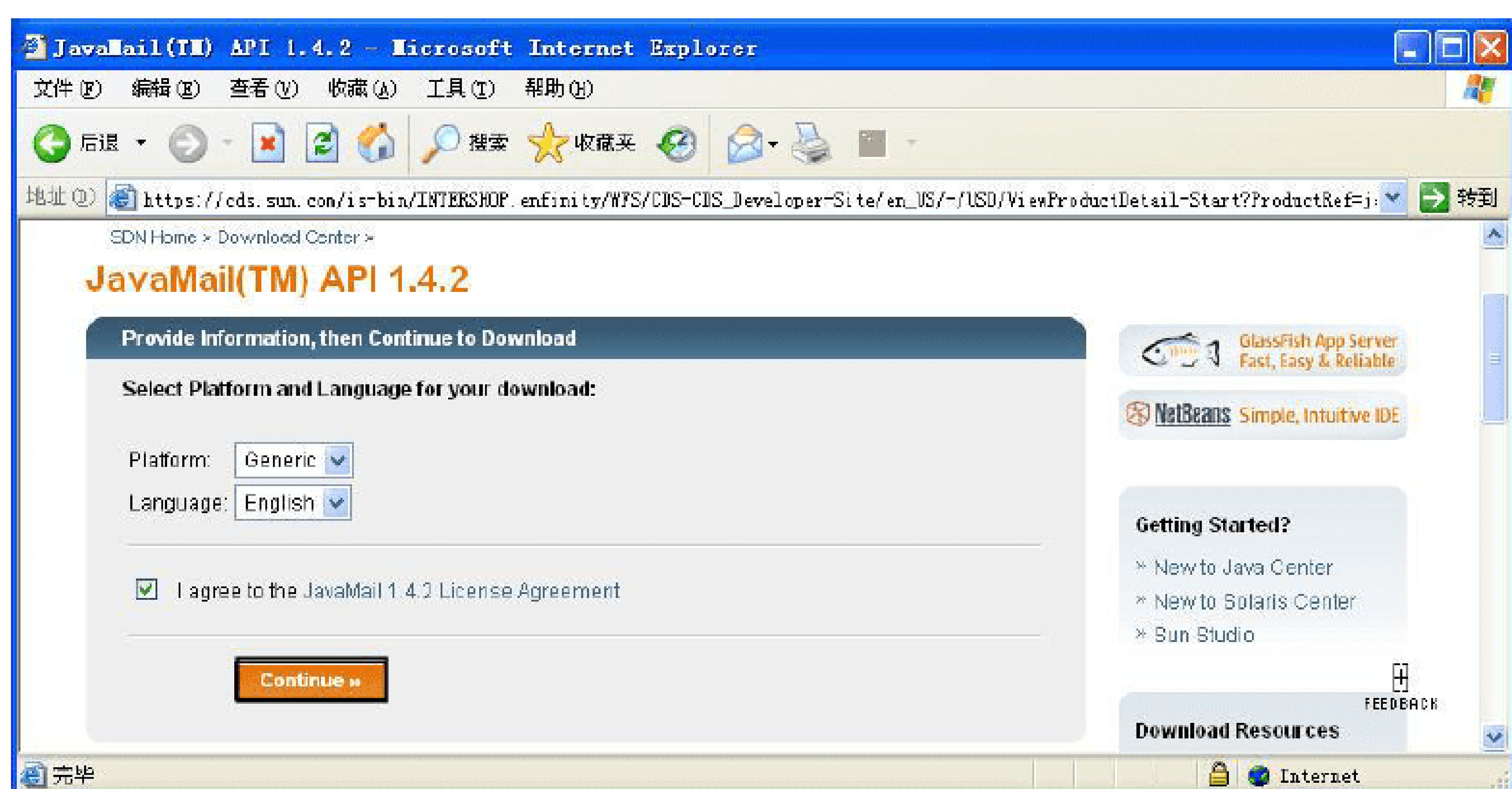


图 11.22　关于 JavaMail 下载页面

（5）在下载 JavaMail 的真正页面（如图 11.23 所示）中，单击 javamail-1.4.2.zip 链接就可以实现 JavaMail 的下载。

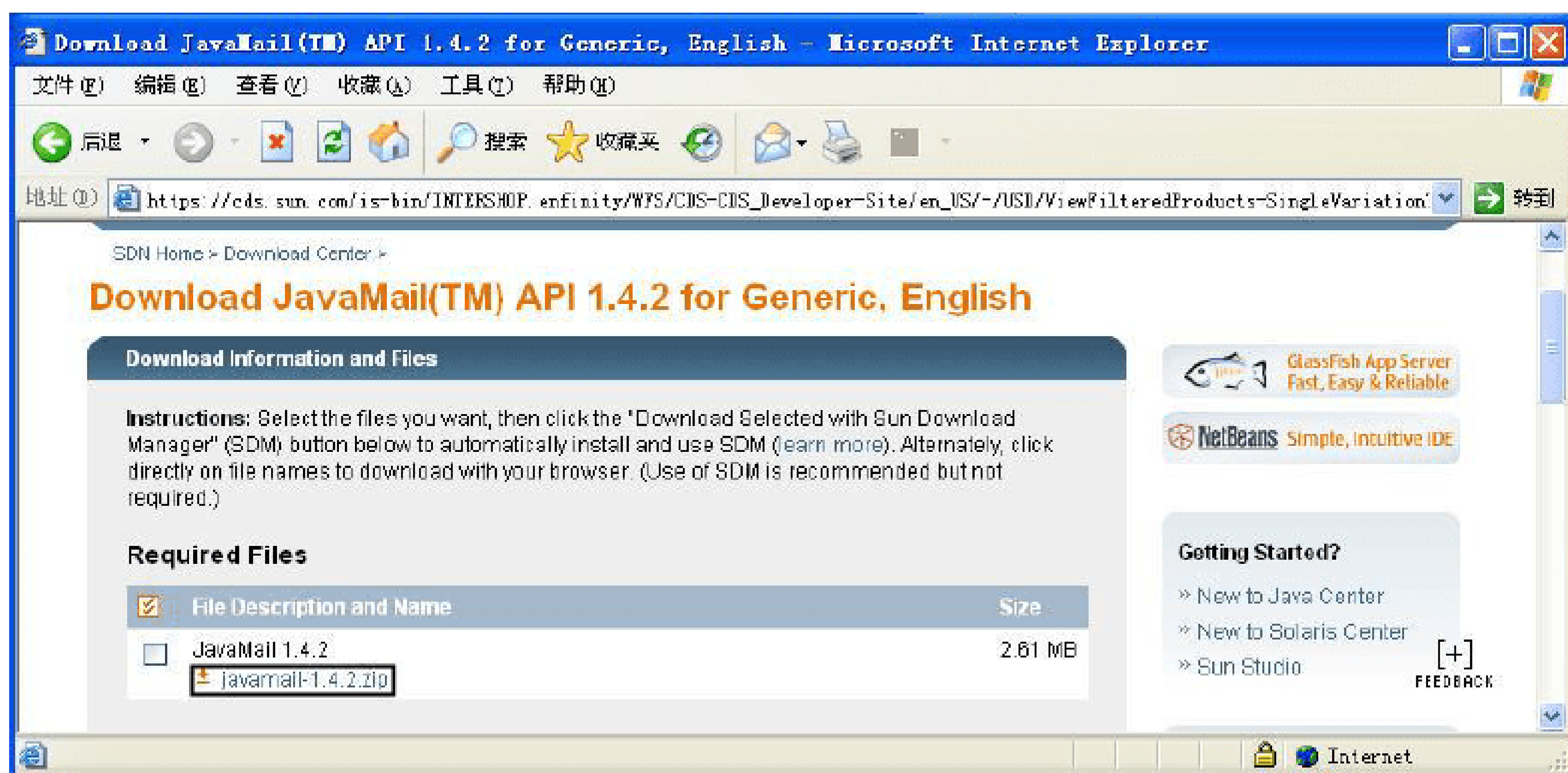


图 11.23　下载 JavaMail

至此，就完成对 JavaMail 组件的下载。

11.2.2　下载 JAF 组件

JAF 组件全称 JavaBeans Activation Framework，其为 Sun 项目组开发的一个辅助 JavaMail 组件的 API，该组件使得 JavaMail 组件操作邮件更简单、容易。目前最新的版本为 JAF1.1.1，具体下载步骤如下：

（1）首先访问 Sun 的官方网站（http://www.java.sun.com/），如图 11.24 所示。在该页面中选择 Products|Java SE 选项就可以进入 Sun 公司中关于 Java SE 产品的页面。

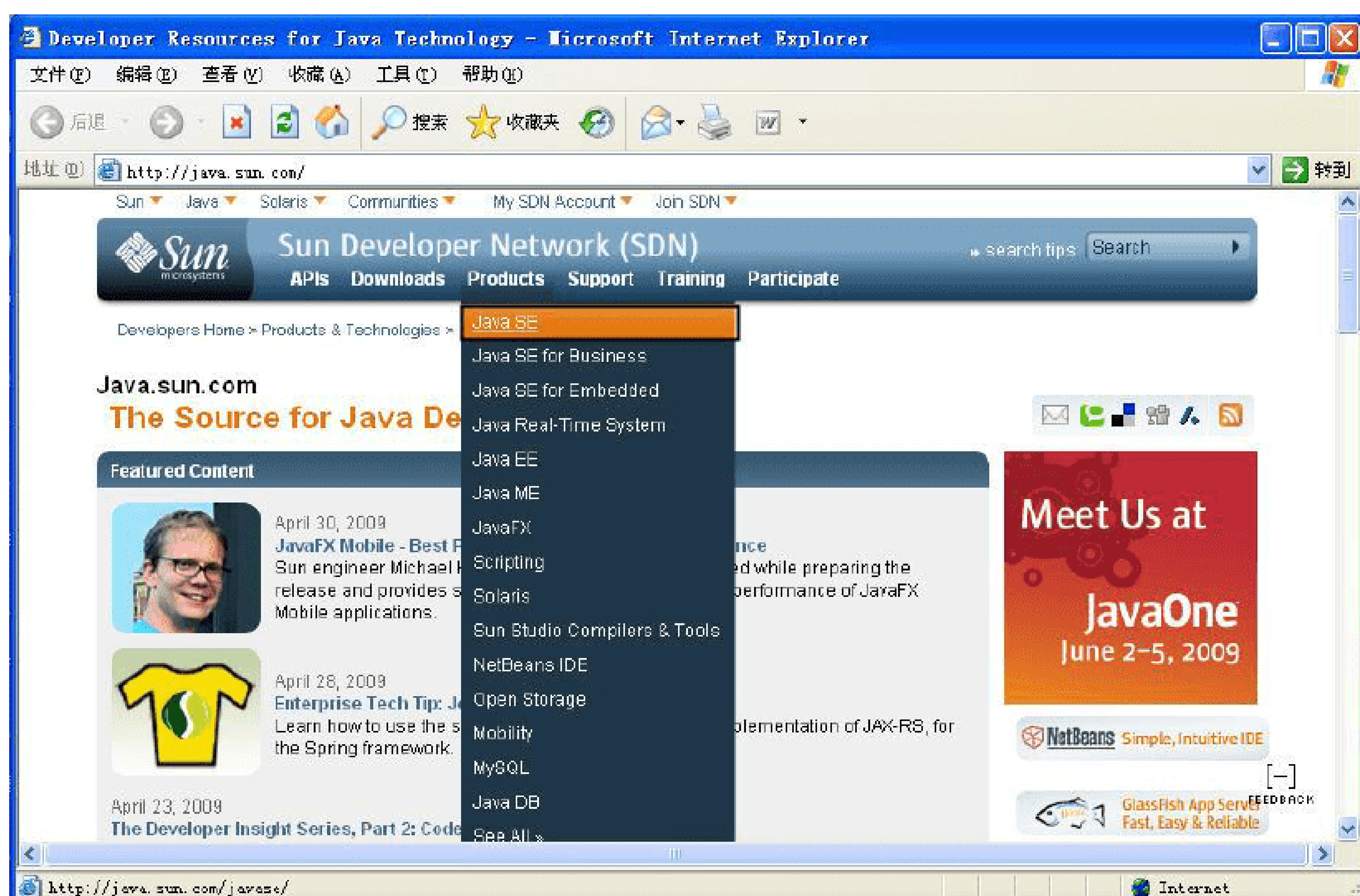


图 11.24　Sun 官方首页

（2）在关于 Java SE 产品的页面中，选择 Technologies|Desktop 命令就会进入关于桌面技术的页面(如图 11.25 所示)。在该页面的下面单击 JavaBeans 链接就会进入关于 JavaBeans 技术的页面（如图 11.26 所示）。

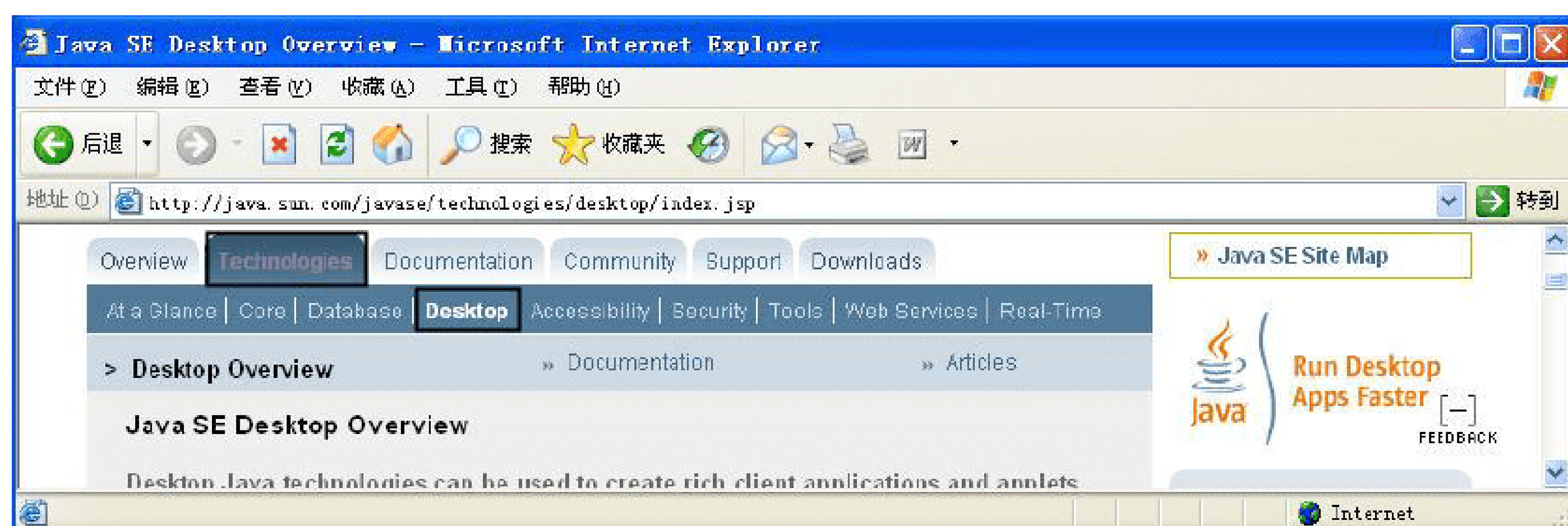


图 11.25　关于桌面技术的页面

（3）关于 JavaBeans 技术的页面中（如图 11.27 所示），单击 Refernece 选项下的 API’s 链接就可以进入关于 Sun 公司 API 的页面（如图 11.28 所示）。在该页面右边 Products & Technologies 目录中单击 JavaBeans Activation Framework 选项就可以进入关于 JAF 组件的页面。

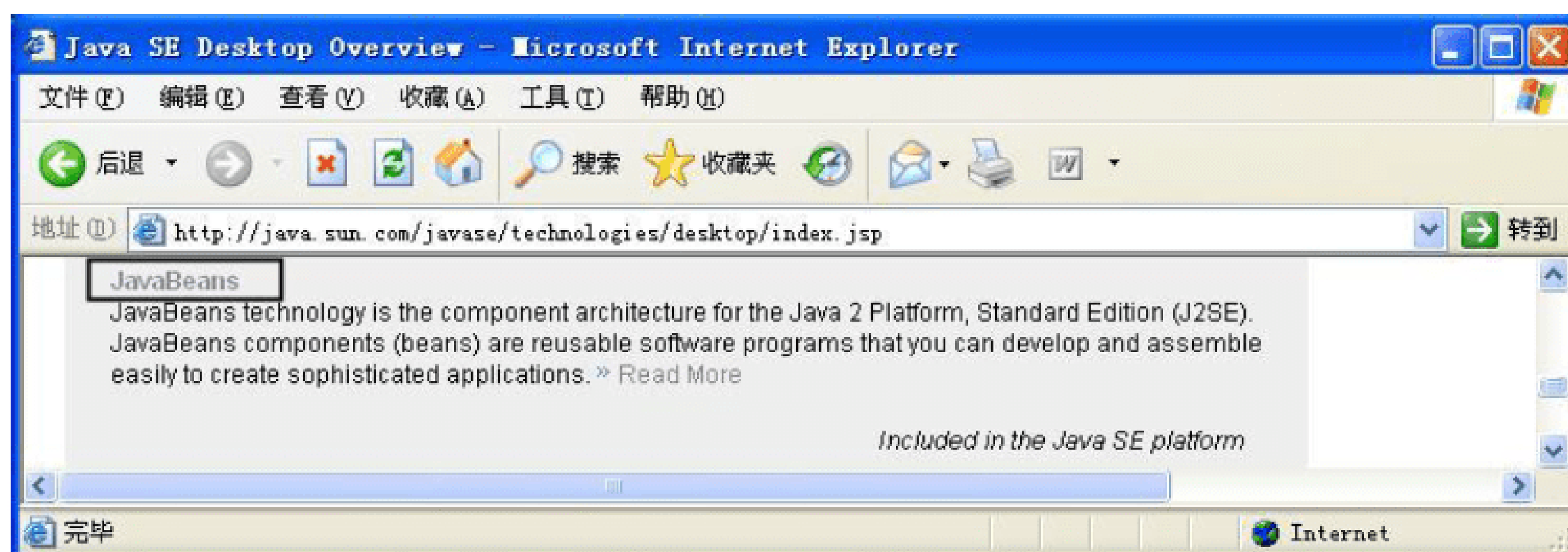


图 11.26　选择 JavaBeans 选项

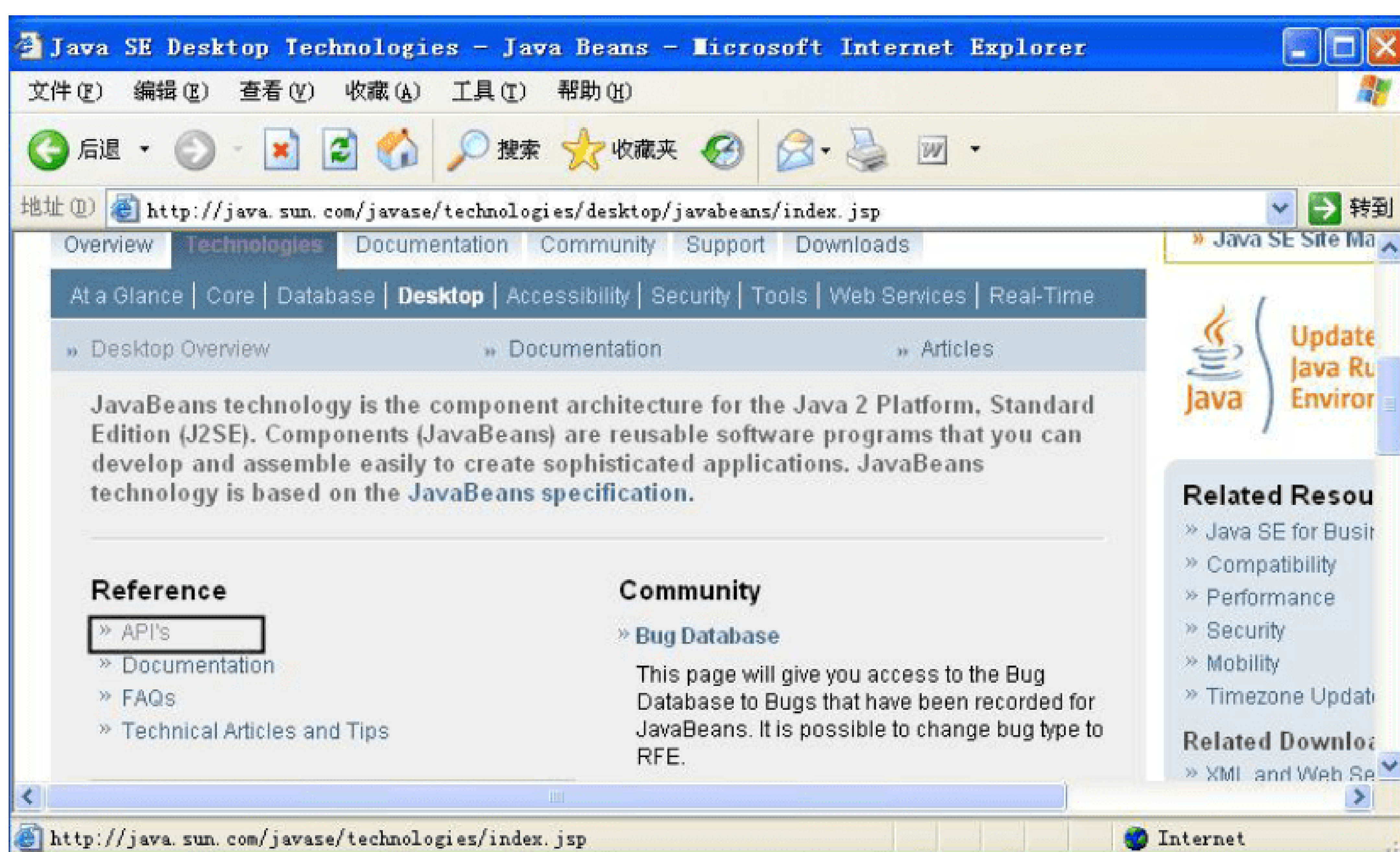


图 11.27　选择 API 选项

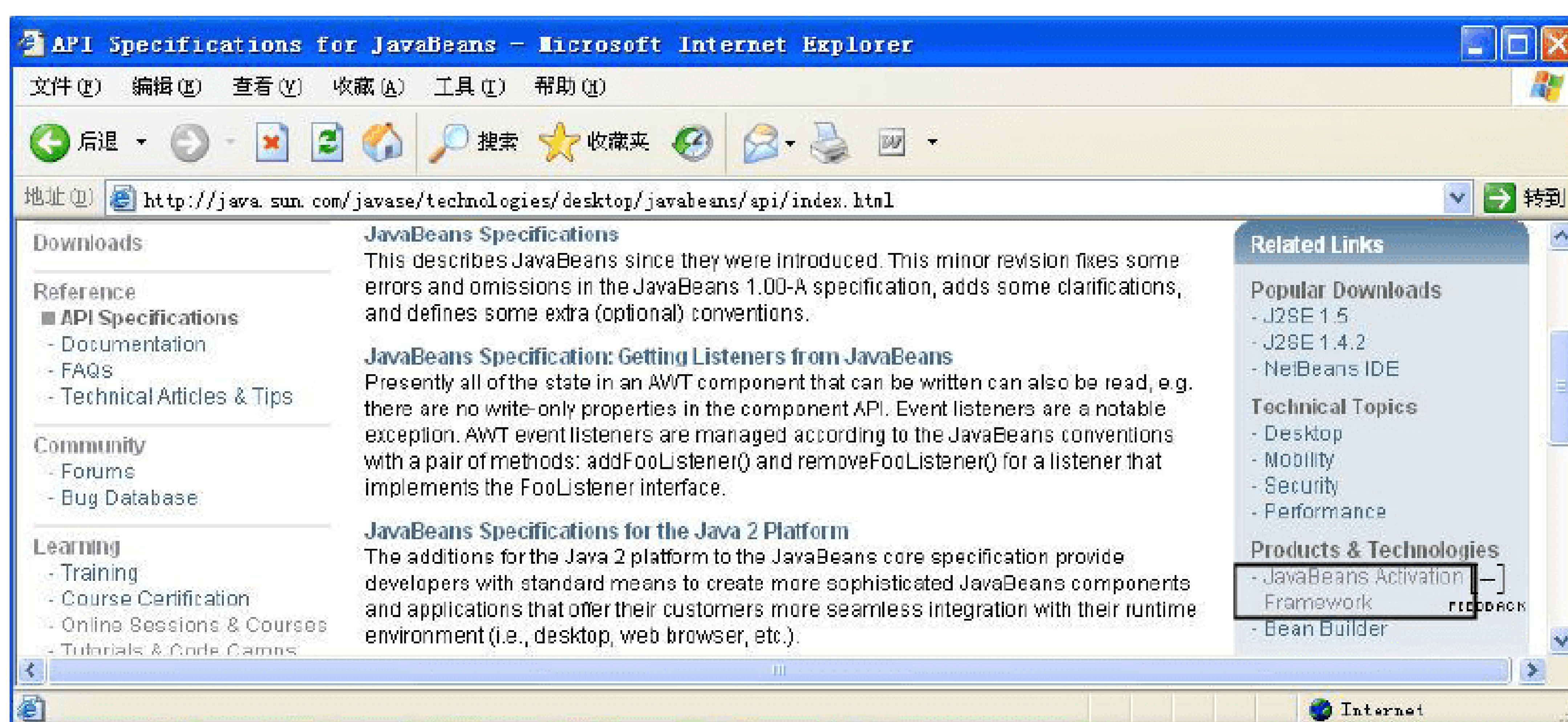


图 11.28　选择 JAF 组件选项

（4）在关于 JAF 组件的页面中（如图 11.29 所示），单击右边 Related Links 目录中的 JAF1.1.1 链接就可以进入关于该组件下载的页面。

（5）在关于 JAF1.1.1 组件下载的页面中（如图 11.30 所示），单击 Download 链接就可以进入下载该组件的真正页面（如图 11.31 所示）。在该页面中，单击 jaf-1_1_1.zip 链接就可实现对该组件的下载。

至此，就完成了对 JAF 组件的下载。

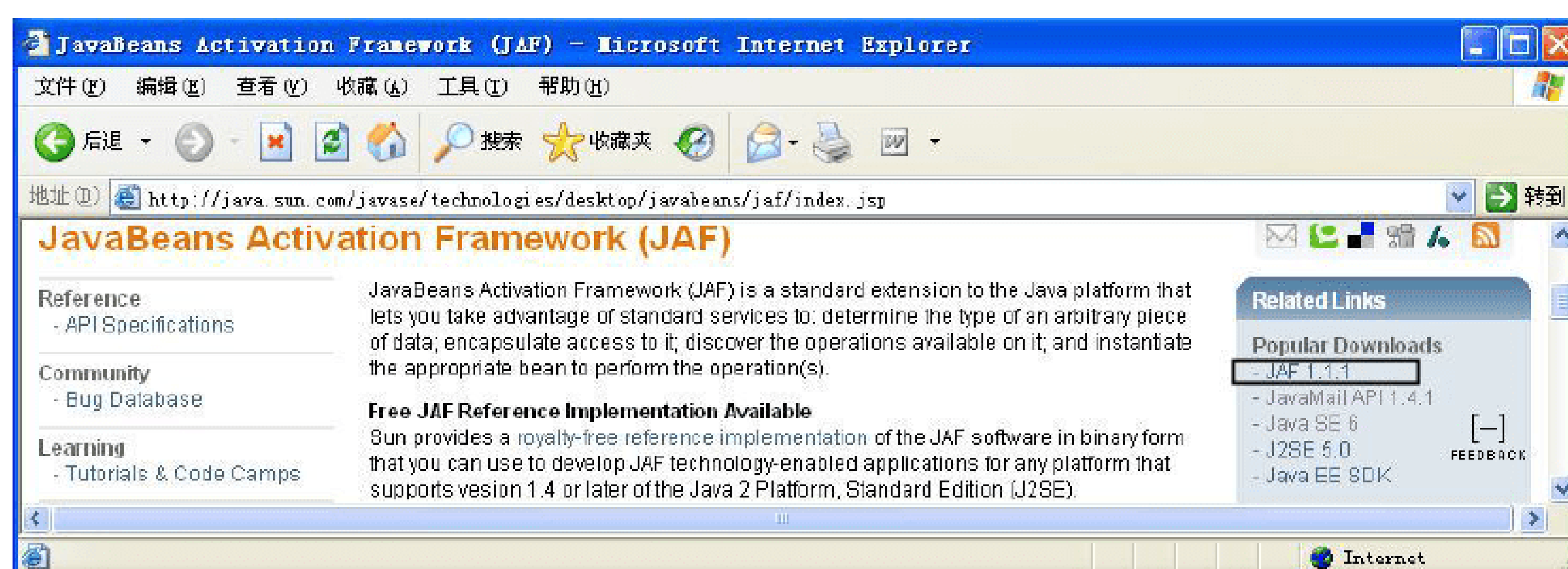


图 11.29　选择 JAF1.1.1

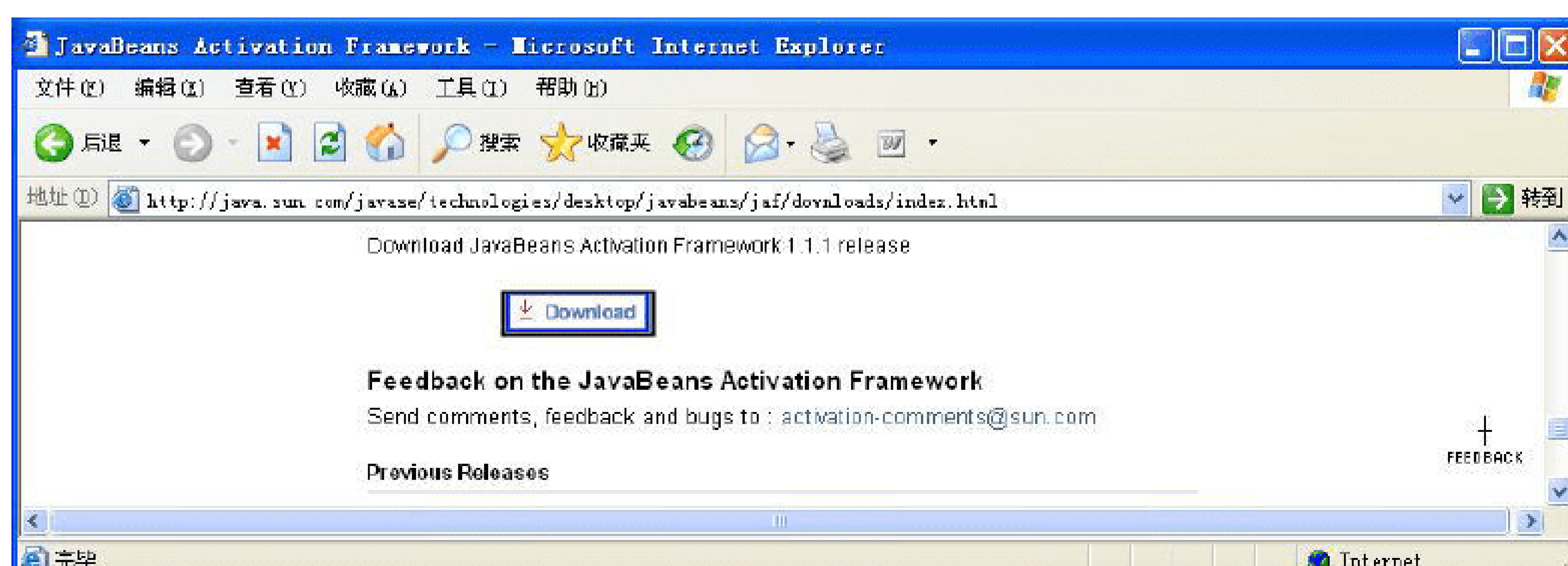


图 11.30　关于下载 JAF1.1.1 组件页面

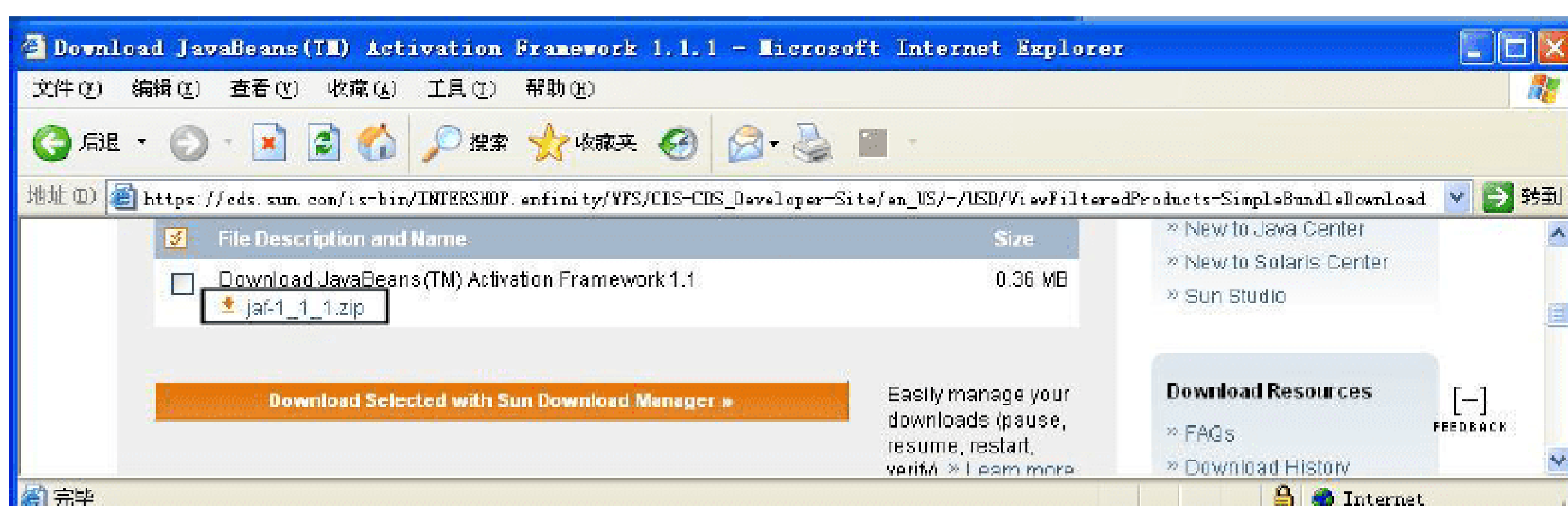


图 11.31　下载 JAF 组件

11.2.3　下载 Cos 上传文件组件

当实现上传文件到服务器时，可以使用 JspSmart 公司的 JspSmartupload 组件、O`rilly 公司的 Cos 组件、Jakarta Apache 公司的 Commonsfileupload 组件、JavaZoom 公司的 uploadbean 组件，更有 struts 组件中自带的 org.apache.struts.upload 类工具等。本章将使用 Cos 组件实现文件上传功能，该组件的下载步骤如下。

（1）首先访问 Servlets 的官方网站（http://www.servlets.com/），如图 11.32 所示。在该页面中单击 New Cos Release 链接就可以转入关于 Cos 组件的页面。

（2）在关于 Cos 组件的页面中（如图 11.33 所示），单击 New Cos Release 标题里的链接地址，就可以转到关于 Cos 组件下载的页面。

（3）在关于 Cos 组件的下载页面中（如图 11.34 所示），单击 cos-26Dec2008.zip 链接就可以完成该组件的下载。

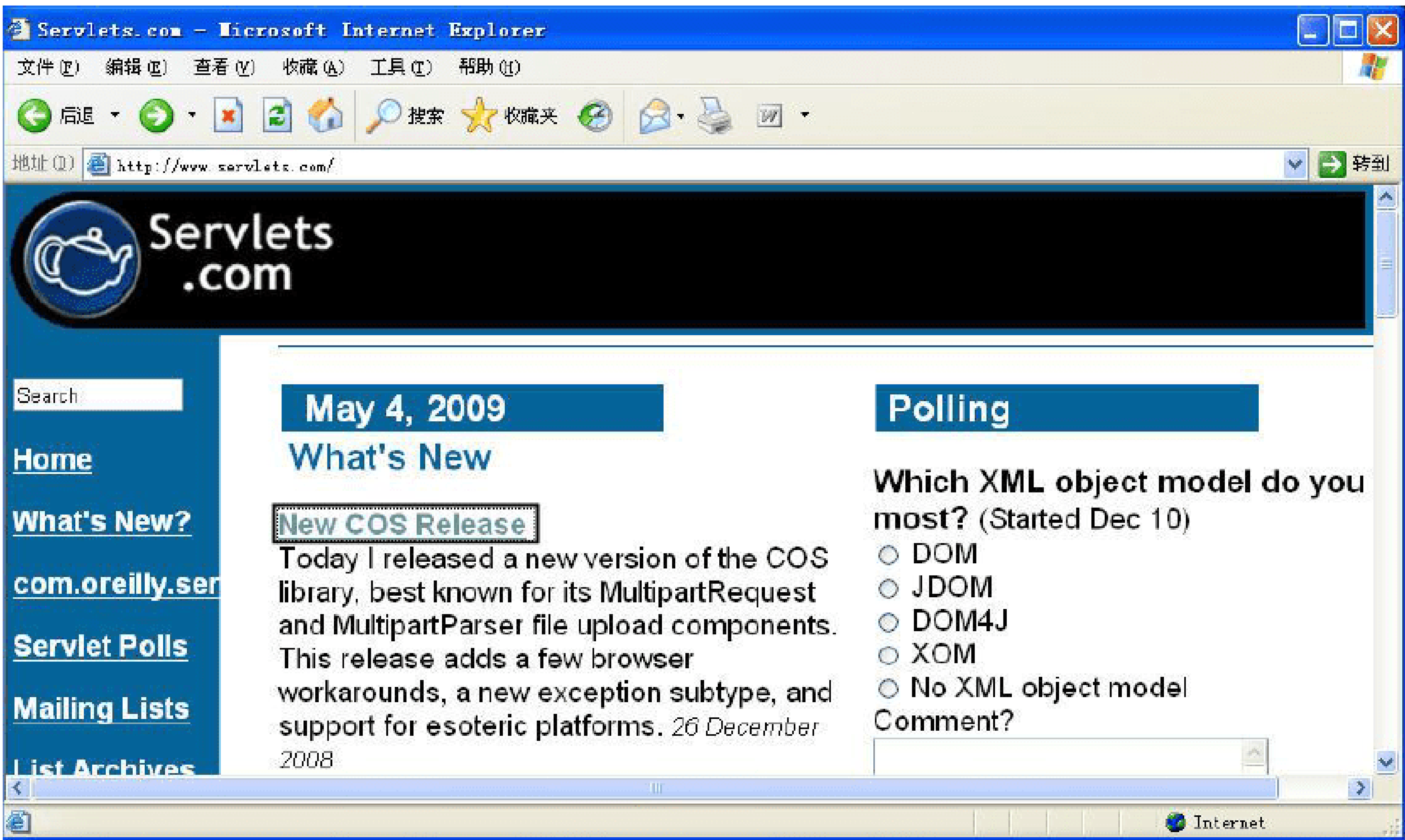


图 11.32　Servlets 首页

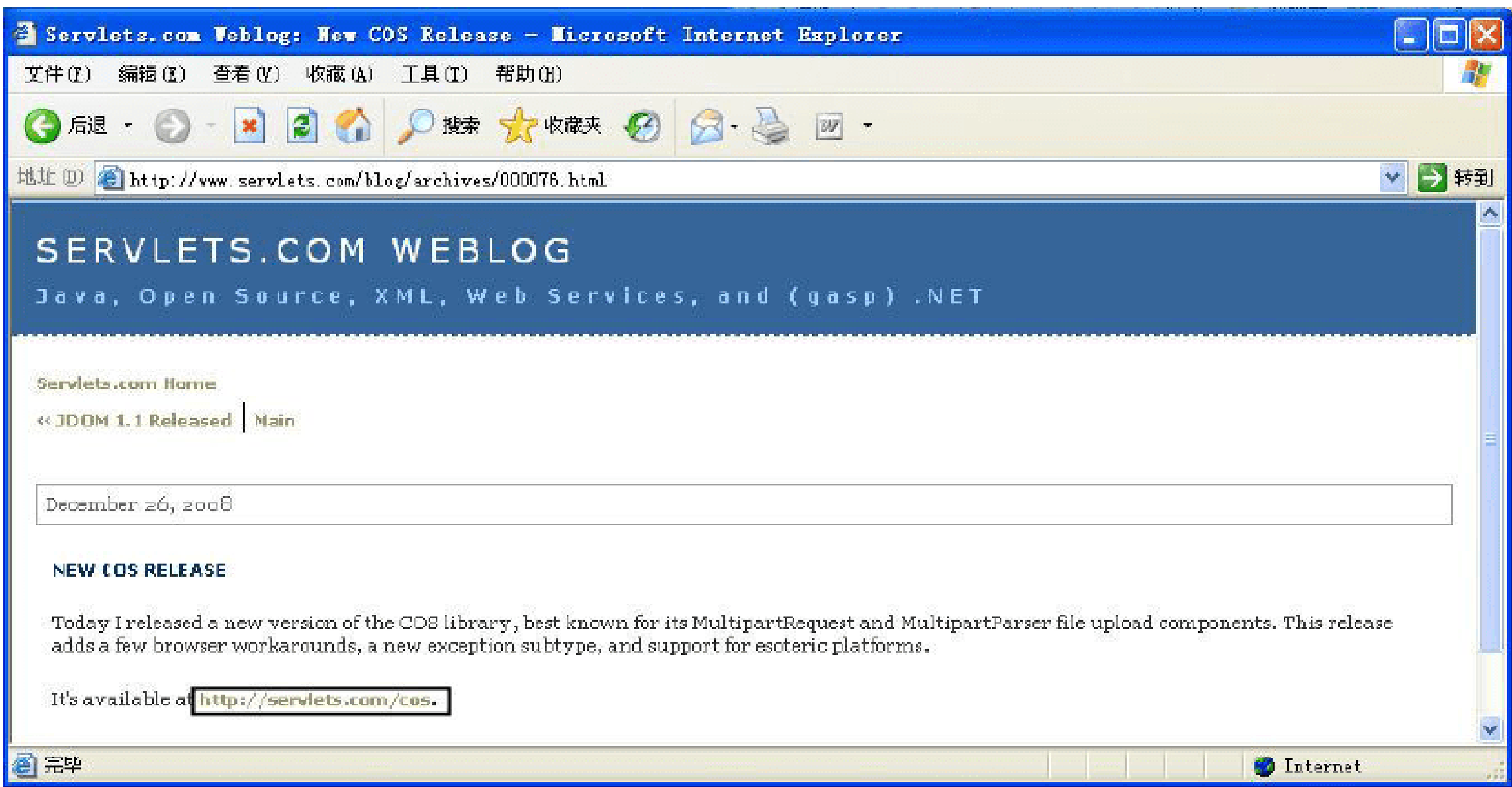


图 11.33　关于 Cos 的页面

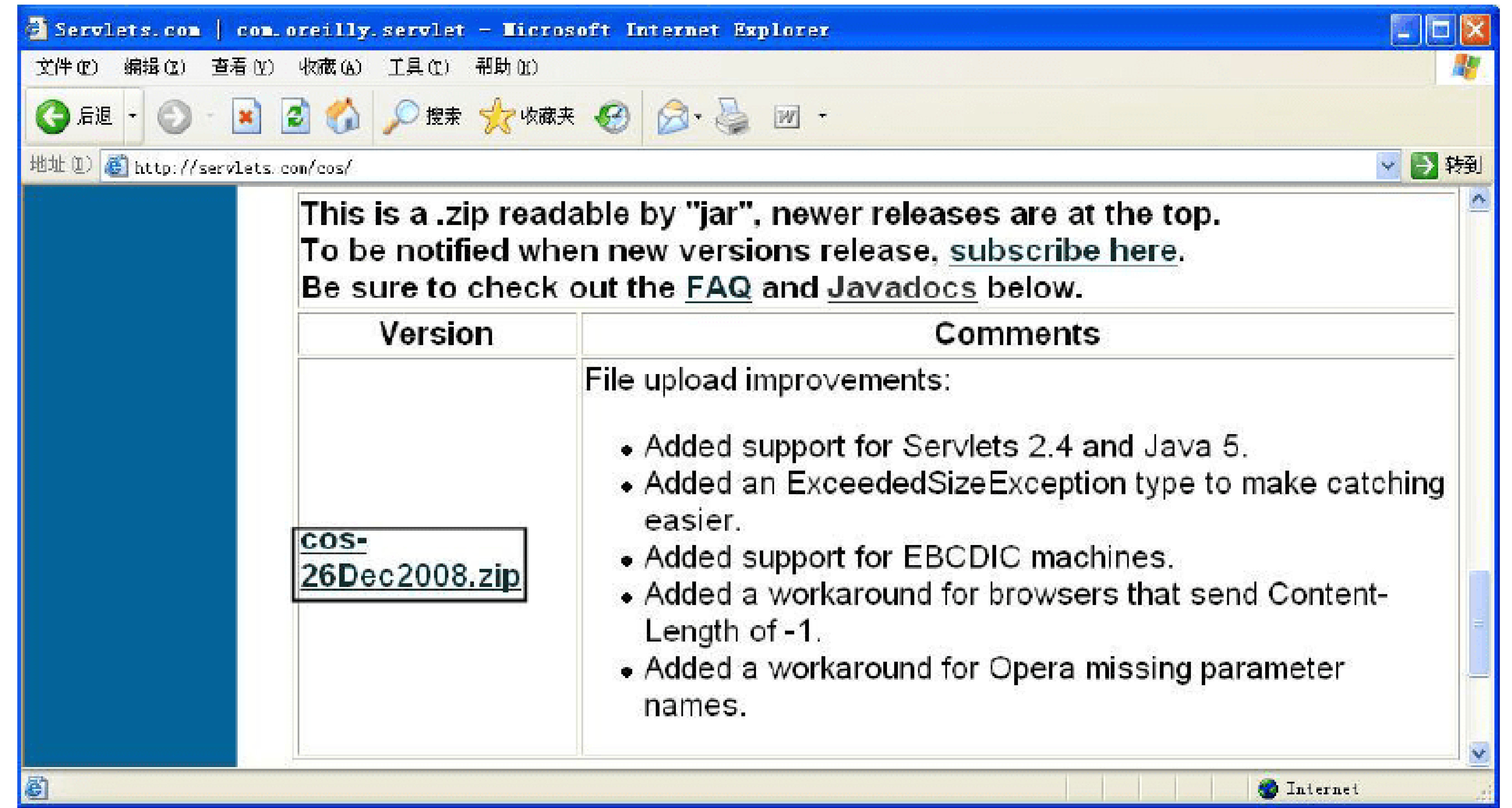


图 11.34　下载 Cos 的 jar 包

至此，就完成对 Cos 组件的下载。

11.3　普通方式电子邮件的发送

本章将通过 JSP+Servlet+JavaBean 框架技术来实现普通电子邮件的发送。普通电子邮件发送程序架构如图 11.35 所示，它包含一个 HTML 页面和一个 Servlet 程序：simplemail.html 和 simplesend.java。

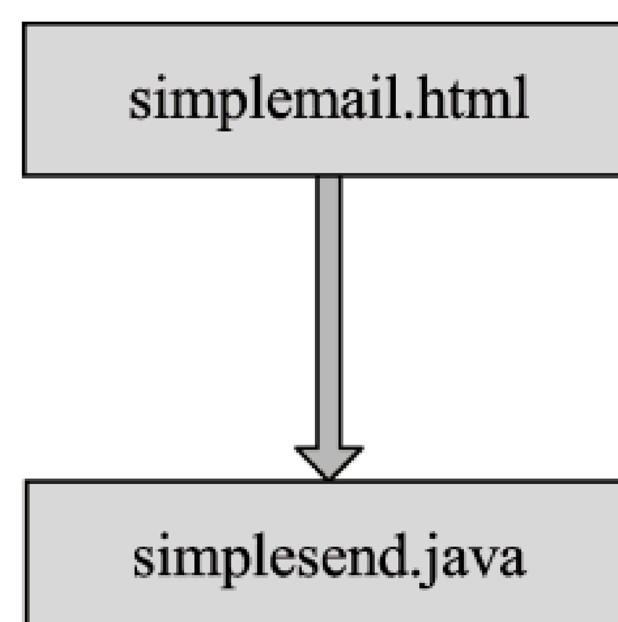


图 11.35　流程图

11.3.1　发送简单邮件页面

simplemail.html 页面用来实现邮件的发送，在该页面中通过在页面的 From 文本框中输入发送邮件的电子邮件地址、TO 文本框中输入要发送邮件的电子邮件地址、Subject 文本框中输入所要发送电子邮件的主题，以及在 Context 文本框中输入所要发送的邮件内容来完成邮件的发送。该页面的具体内容如代码 11.1 所示。

代码 11.1　发送简单邮件页面：simplemail.html

```
...
    <body>
        <form action="send  simplemail" method="post">
            From:
            <input name="from">
            <br>
            TO:
            <input name="to">
            <br>
            Subject:
            <input name="subject">
            <br>
            Context:
            <textarea rows="40" cols="30" name="context"></textarea>
            <br>
            <input type="submit">
        </form>
    </body>
...
```

注意： 在上述代码中，根据标签<form>的 action 属性可以发现提交的信息由名为 simplesend 的 Servlet 程序来处理。

11.3.2　处理发送简单邮件

simplesend.java 文件是服务器端程序，用来处理由文件 simplemail.html 发送的信息。该文件的具体内容如代码 11.2 所示。

代码 11.2　处理简单邮件：simplesend.java

```
...
```

```
public class simplesend extends HttpServlet {
    //编辑 doPost()方法
    public  void  doPost(HttpServletRequest  request,  HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        request.setCharacterEncoding("gb2312");          //设置编码方式
        String from = request.getParameter("from");      //获取 From 内容
        String to = request.getParameter("to");          //获取 To 内容
        String subject = request.getParameter("subject");
                                                         //获取 Subject 内容
        String context = request.getParameter("context");
                                                         //获取 Context 内容
        String mailserver = "d9e307714c5044f"; //确定要发送的邮件服务器的地址
        //设置邮件的传输协议
        try {
            Properties prop = System.getProperties();
            prop.put("mail.smtp.host", mailserver);
            //建立邮件发送的连接
            Session session = Session.getDefaultInstance(prop, null);
            Message msg = new MimeMessage(session);     //创建发送的信息的载体
            msg.setFrom(new InternetAddress(from));    //设置相关的邮件属性
            //点到点的发送
            msg.setRecipient(Message.RecipientType.TO, new Internet
            Address(to));
            msg.setSubject(subject);
            msg.setSentDate(new Date());
            msg.setText(context);
            Transport.send(msg);                         //发送
        } catch (Exception e) {
        }
        out.print("send ok");                            //发送成功
        out.flush();
        out.close();
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 simplesend 类路径-->
<servlet>
  <servlet-name>simplesend</servlet-name>
  <servlet-class>com.cjg.servlet.simplesend</servlet-class>
</servlet>
<!--配置 simplesend 类映射路径-->
<servlet-mapping>
  <servlet-name>simplesend</servlet-name>
  <url-pattern>/simplesend</url-pattern>
</servlet-mapping>
```

【代码解析】

- ❑ 首先获取请求的所有参数。
- ❑ 接着设置邮件的传输协议，在具体步骤中首先通过 Properties 类存储关于邮件连接信息，然后获取关于 Properties 类的 Session 对象，最后通过以 Session 对象为参数的 Message 类创建出关于邮件的载体 msg。

- 最后，通过 Transport.send()方法发送设置好相关属性的 msg 对象。

11.4　HTML 方式电子邮件的发送

本章将通过 JSP+Servlet+JavaBean 框架技术来实现 HTML 方式电子邮件的发送，HTML 方式电子邮件的发送程序架构如图 11.36 所示，它包含一个 HTML 页面和一个 Servlet 程序：htmlmail.html 和 HtmlSend.java。

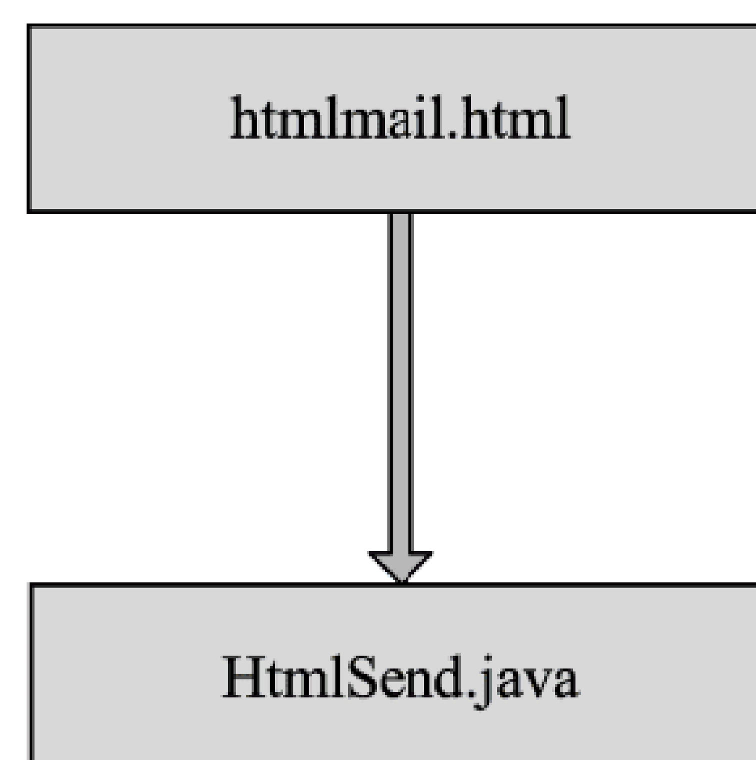


图 11.36　程序关系图

11.4.1　发送选择类型邮件页面

htmlmail.html 页面用来实现邮件的发送，在该页面中通过在页面的 From 文本框中输入发送邮件的电子邮件地址、TO 文本框中输入要发送邮件的电子邮件地址、Subject 文本框中输入所要发送电子邮件的主题、Context 文本框中输入所要发送的邮件的内容，以及 type 选择框决定显示内容的方式实现邮件的发送。该页面的具体内容如代码 11.3 所示。

代码 11.3　发送简单邮件页面：htmlmail.html

```
...
    <body>
        <form action="SEND1" method="post">
            From:
            <input name="from">
            <br>
            TO:
            <input name="to">
            <br>
            Subject:
            <input name="subject">
            <br>
            type:
            <select name=type size="1">
                <option value="text/plain">Text</option>
                <option value="text/html">Html</option>
            </select>
            Context:
            <textarea rows="40" cols="30" name="context"></textarea>
            <br>
            <input type="submit">
        </form>
    </body>
...
```

注意：在上述代码中，该文件代码比 simplemail.html 文件多出了一个类型选择框，通过对该选择框相应类型的选择，可以决定接收者查看的页面内容。

11.4.2　处理发送各种类型邮件

HtmlSend.java 文件是服务器端程序，用来处理由文件 htmlmail.html 发送的信息。该文件的具体内容如代码 11.4 所示。

代码 11.4　处理各种类型邮件：HtmlSend.java

```
...
public class HtmlSend extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        request.setCharacterEncoding("gb2312");
        String from = request.getParameter("from");
        String to = request.getParameter("to");
        String subject = request.getParameter("subject");
        String context = request.getParameter("context");
        String type=request.getParameter("type");
        //确定要发送的邮件服务器的地址
        String mailserver = "d9e307714c5044f";
        //设置邮件的传输协议
        try {
            Properties prop = System.getProperties();
            prop.put("mail.smtp.host", mailserver);
            //建立邮件发送的连接
            Session session = Session.getDefaultInstance(prop, null);
            //创建发送的信息载体
            Message msg = new MimeMessage(session);
            //设置相关的邮件属性
            msg.setFrom(new InternetAddress(from));
            //点到点的发送
            msg.setRecipient(Message.RecipientType.TO, new Internet
            Address(to));
            msg.setSubject(subject);
            msg.setSentDate(new Date());
            //判断发送的 Mime 类型
            Multipart mp=new MimeMultipart();       //创建邮件的信仁对象
            MimeBodyPart mbp=new MimeBodyPart();    //创建邮件的信仁对象部件
            mbp.setContent(context, type+";charset=GB2312");
                                                    //设置邮件部件的类型
            mp.addBodyPart(mbp);                    //添加 mbp 对象到 mp 对象
            msg.setContent(mp);                     //添加 mp 对象到 msg 对象
            Transport.send(msg);                    //发送
        } catch (Exception e) {
        }
        out.print("send ok");
        out.flush();
        out.close();
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
    <!--配置 HtmlSend 类路径-->
    <servlet>
        <servlet-name>HtmlSend</servlet-name>
        <servlet-class>com.cjg.servlet.HtmlSend</servlet-class>
    </servlet>
    <!--配置 HtmlSend 类映射路径-->
    <servlet-mapping>
        <servlet-name>HtmlSend</servlet-name>
        <url-pattern>/HtmlSend</url-pattern>
    </servlet-mapping>
```

【代码解析】

在上述代码中除了获取设置好的 Message 对象 msg 后，最后还需要根据邮件的类型进行相应的类型显示。如何设置邮件内容的类型呢？首先需要创建 Multipart 类对象 mp 作为信仁和 MimeBodyPart 类对象 mbp 作为信仁部件，然后通过 mbp.setContent()方法设置邮件部件的类型，最后添加信仁部件到信仁。

11.5　携带附件电子邮件的发送

本章将通过 JSP+Servlet+JavaBean 框架技术来实现携带附件方式电子邮件的发送。携带附件方式电子邮件发送程序架构如图 11.37 所示，它包含一个 HTML 页面和一个 Servlet 程序：filemail.html 和 FileSend.java。

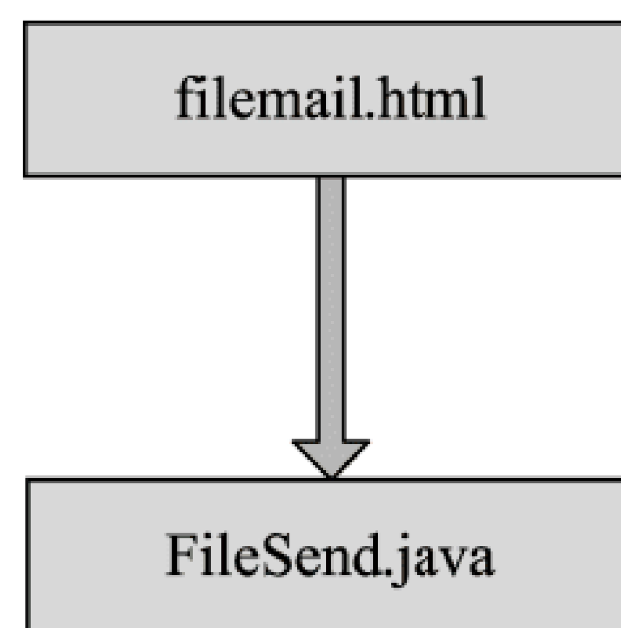


图 11.37　程序关系图

11.5.1　发送携带附件邮件页面

filemail.html 页面用来实现邮件的发送，在该页面中通过在页面的 From 文本框中输入发送邮件的电子邮件地址、TO 文本框中输入要发送邮件的电子邮件地址、Subject 文本框中输入所要发送电子邮件的主题、Context 文本框中输入所要发送的邮件的内容，以及 file 上传文件按钮实现文件的上传。该页面的具体内容如代码 11.5 所示。

代码 11.5　发送携带附件邮件页面：filemail.html

```
...
    <body>
        <!--设置 form 标签相关属性-->
        <form action="SEND2" method="post" enctype="multipart/form-data">
            From:
            <input name="from">                <!--From 文本框-->
            <br>
            TO:
            <input name="to">                  <!--To 文本框-->
            <br>
            Subject:
            <input name="subject">             <!--Subject 文本框-->
            <br>
```

```
            type:
            <select name=type size="1">    <!--设置选择框--->
                <option value="text/plain">Text</option>
                <option value="text/html">Html</option>
            </select>
            //设置 file 文件标签
            file:<input type="file" name="filename">
            Context:
            <textarea rows="20" cols="30" name="context"></textarea>
            <br>
            <input type="submit">
        </form>
    </body>
...
```

【代码解析】

- 为了能够实现携带附件，在设置标签<form>时，属性 method 的值必须为 post，而属性 enctype 的值必须为 multipart/form-data。这是因为携带附件实际上就是实现文件的上传功能。
- 为了能够实现对携带附件的选择，在该页面中引入 fiile 标签。

11.5.2　处理发送附件邮件

FileSend.java 文件是服务器端程序，用来处理由文件 filemail.html 发送的信息。该文件的具体内容如代码 11.6 所示。

代码 11.6　处理携带附件邮件：FileSend.java

```
...
public class FileSend extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=gb2312");
                                                            //设置响应消息头
        PrintWriter out = response.getWriter();             //获取输出流
        request.setCharacterEncoding("gb2312");             //设置编码方式
        //利用 oreilly 对象重新封装请求对象
        MultipartRequest req=new MultipartRequest(request,".",5*1024*
        1024,"GB2312");
        //获取请求参数中的所有的参数
        String from = req.getParameter("from");
        String to = req.getParameter("to");
        String subject = req.getParameter("subject");
        String context = req.getParameter("context");
        String type=req.getParameter("type");
        String filename=req.getFilesystemName("filename");    //获取文件名
        String mailserver = "d9e307714c5044f";
                                            //确定要发送的邮件服务器地址
        //设置邮件的传输协议
        try {
            Properties prop = System.getProperties();
            prop.put("mail.smtp.host", mailserver);
            Session session = Session.getDefaultInstance(prop, null);
```

```
                                                   //建立邮件发送的连接
            Message msg = new MimeMessage(session);  //创建发送的信息的载体
            msg.setFrom(new InternetAddress(from));   //设置相关的邮件属性
            //点到点的发送
            msg.setRecipient(Message.RecipientType.TO, new Internet
            Address(to));
            //处理文件的上传
            if(filename!=null){
                //File file=new File(filename);//存留备份
                MimeBodyPart mbp1=new MimeBodyPart();
                mbp1.setContent(context, type+";charset=GB2312");
                                                   //对邮件普通信息的处理
                //附件的处理
                MimeBodyPart mbp2=new MimeBodyPart();
                FileDataSource fds=new FileDataSource(filename);
                                                          //设置数据源
                mbp2.setDataHandler(new DataHandler(fds)); //封装数据源
                //通过 encodeText()方法封装到 MimeBodyPart 对象中
                mbp2.setFileName(MimeUtility.encodeText(fds.getName(),
                "GB2312","B"));
                Multipart mp=new MimeMultipart();
                mp.addBodyPart(mbp1);
                mp.addBodyPart(mbp2);
                msg.setContent(mp);
            }
            else{
                msg.setContent(context, type+";charset=GB2312");
                                                          //普通邮件的处理
            }
            Transport.send(msg);                          //实现发送
        } catch (Exception e) {
        }
        out.print("send ok"+"<br>");
        out.print("<a href='http://localhost/mail/index.asp'>查看信件
        </a>");
        out.flush();
        out.close();
    }
}
```

接着在 web.xml 文件中对该 Servlet 程序进行配置。

```
<!--配置 FileSend 类路径-->
<servlet>
  <servlet-name>FileSend</servlet-name>
  <servlet-class>com.cjg.servlet.FileSend</servlet-class>
</servlet>
<!--配置 FileSend 映射路径-->
<servlet-mapping>
  <servlet-name>FileSend</servlet-name>
  <url-pattern>/FileSend</url-pattern>
</servlet-mapping>
```

【代码解析】

- ❑ 首先利用 oreilly 技术把所有的请求封装到类 MultipartParser 中，之所以这样是因为 MultipartParser 类是一个可以处理 multipart/form-data 类型请求参数的类，该类

可以把上传的文件以二进制的形式存储到输出流中。

- 在具体使用 MultipartRequest 类时，第 1 个参数为请求对象；第 2 个参数之所以为“.”，是因为每个文件名中都会有该符号；第 3 个参数用来设置上传文件大小的最大值；当第 4 个参数设置为 GB2312 时，则可以上传文件名为中文的文件。

11.6　多学两招——关于邮件的基础知识

为了让读者能够顺利地开发其他基于邮件功能的模块，本节将详细讲解关于邮件的基础知识和基本原理。关于邮件知识主要有两方面内容：发送、接收邮件和创建、解析邮件内容。即在发送邮件之前，首先要创建出邮件；在接收到邮件后，如果想查看邮件内容必须要解析邮件。

11.6.1　关于邮件的基础概念

本节将详细讲解在开发基于邮件功能时经常遇到的概念：电子邮件服务器、电子邮件传输协议、电子邮箱、电子邮件客户端软件和电子邮件传输过程。

1．电子邮件服务器

在现在的 Internet 上存在大量的电子邮件服务器，例如国内的 163.com、sohu.com 等网站都提供了免费的面向公众的电子邮件服务器。在许多大的公司里也提供了面向内部员工的电子邮件服务器。不管是面向公众的电子邮件服务器，还是面向公司内部的电子邮件服务器，都可以实现电子邮件的发送和接收。

电子邮件服务器跟现实生活中的邮局类似，可以实现如下功能：

- 接收用户投递的电子邮件。
- 接收其他电子邮件服务器转发过来的电子邮件。
- 将用户投递的电子邮件发送给目标电子邮件服务器。
- 用户读取其他用户发送给他自己的电子邮件。

电子邮件的传输过程跟现实生活中信件的传输过程一样（如图 11.38 所示）。首先用户把信投递到当地的邮局，然后该邮局根据收信人的地址投递到目标邮局。同时当地邮局还接收发给当地用户的邮件，最后用户去邮局取发给自己的信。



图 11.38　电子邮件简单传输过程

电子邮件服务器按照通信协议可以划分为两种类型：

- SMTP 服务器：该服务器相当于现实生活邮局的邮件接收部门，即负责为用户发送邮件，接受其他服务器发送给用户的邮件。
- POP3/IMAP 服务器：该服务器相当于现实生活中邮局里为专门取信服务的部门，即负责为用户读取 SMTP 服务器接收进来关于自己的邮件。

2．电子邮件传输协议

SMTP（Simple Mail Transfer Protocol）即简单邮件传输协议，它是一组用于由源地址到目的地址传送电子邮件的规则。使用该协议可以帮助每台计算机在发送或中转电子邮件时找到下一个目的地。

POP3（Post Office Protocol 3）即邮局协议，它是一组用于将个人计算机连接到 Internet 的邮件服务器和下载电子邮件的协议。使用该协议可以允许用户从服务器上把邮件存储到本地主机（即本地计算机）上，同时删除保存在邮件服务器上的邮件。

IMAP（Interactive Mail Access Protocol）即交互式邮件存取协议，跟 POP3 协议非常相似，主要用来使邮件客户端从邮件服务器上获取邮件的信息、下载邮件等。

注意： POP3 协议与 IMAP 协议的区别是，POP3 协议会把电子邮件下载到本地计算机上保存起来，而 IMAP 协议则把电子邮件保留在邮件服务器上。

通过上述的讲解可以知道，在如图 11.38 所示的电子邮件传输过程中，过程 1、2、3 分别使用 SMTP 协议，而过程 4 却使用 POP3 协议。

3．电子邮箱

每个电子邮件服务器上都可以开设多个电子邮箱，电子邮箱也称之为 E-mail 地址，它类似于现实生活中的通信地址。用户可以通过这个地址接收到其他人发来的电子邮件并向其他人发送电子邮件。电子邮箱的获取需要向邮件服务器进行申请，确切地说，电子邮箱其实就是用户在邮件服务器上申请的一个账户。邮件服务器把接收到的邮件保存到为某个账户所分配的邮箱空间中，用户通过其申请的用户名和密码登录到邮件服务器上查收该地址已收到的电子邮件。

4．电子邮件客户端软件

邮件客户端软件负责与邮件服务器通信，主要用于帮助用户将邮件发送给 SMTP 服务器并从 POP3/IMAP 邮件服务器读取用户的电子邮件。邮件客户端软件通常集邮件编写、发送和接收功能于一体。

5．电子邮件的传输过程

为了能够讲清楚电子邮件的传输过程，下面通过具体的实例来讲解图 11.39 的电子邮件的传输过程，具体步骤如下。

首先在 126 邮件服务器和 163 邮件服务器上分别申请账户：cjgong@126.com 和 cjgong_1@163.com。接着通过 cjgong@126.com 账户向 cjgong_1@163.com 账户发送一封邮件。

拥有 cjgong@126.com 账户的用户首先通过 OutLook 客户端，把电子邮件发送给 126 邮件服务器中的 SMTP 服务器，该邮件服务器发现该电子邮件的目标地址为 163 邮件服务器，所以其就把该电子邮件发送给 163 邮件服务器中的 SMTP 服务器。当 163 邮件服务器中的 SMTP 服务器接收到邮件后，就会把该邮件存储到关于 cjgong_1 账户的空间中。以后，当拥有 cjgong_1 账户的用户通过 OutLook 客户端连接到 163 邮件服务器中的 POP3 服务器

时，该服务器就会到 cjgong_1 账户相对应的私人空间查找邮件，如果存在邮件就会把这些邮件返回给 OutLook 客户端。

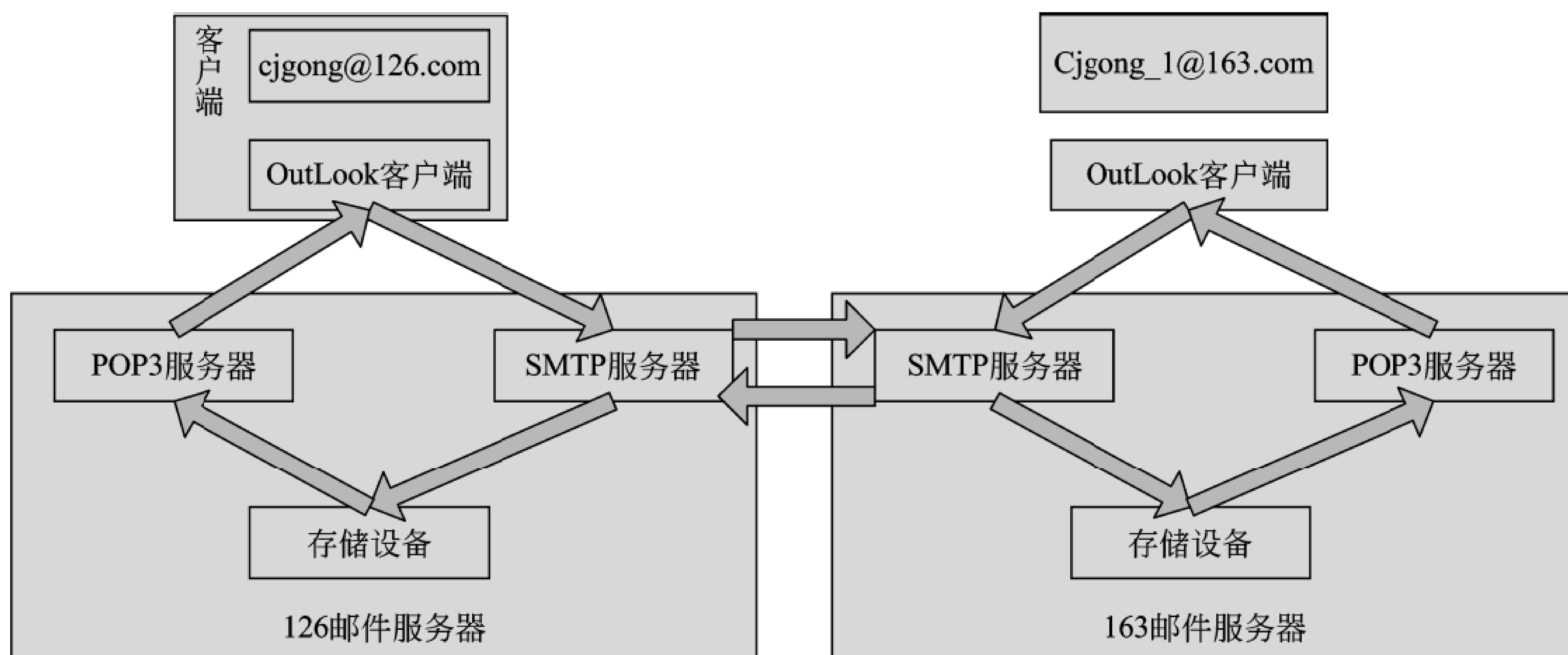


图 11.39　电子邮件的传输过程

至此，就完成了关于邮件相关概念的介绍。

11.6.2　手工发送和接收电子邮件

为了让读者对邮件的基本概念有更深刻的了解，在本节将通过手工的方式来发送、接收一封电子邮件。具体的发送过程如图 11.40 所示。

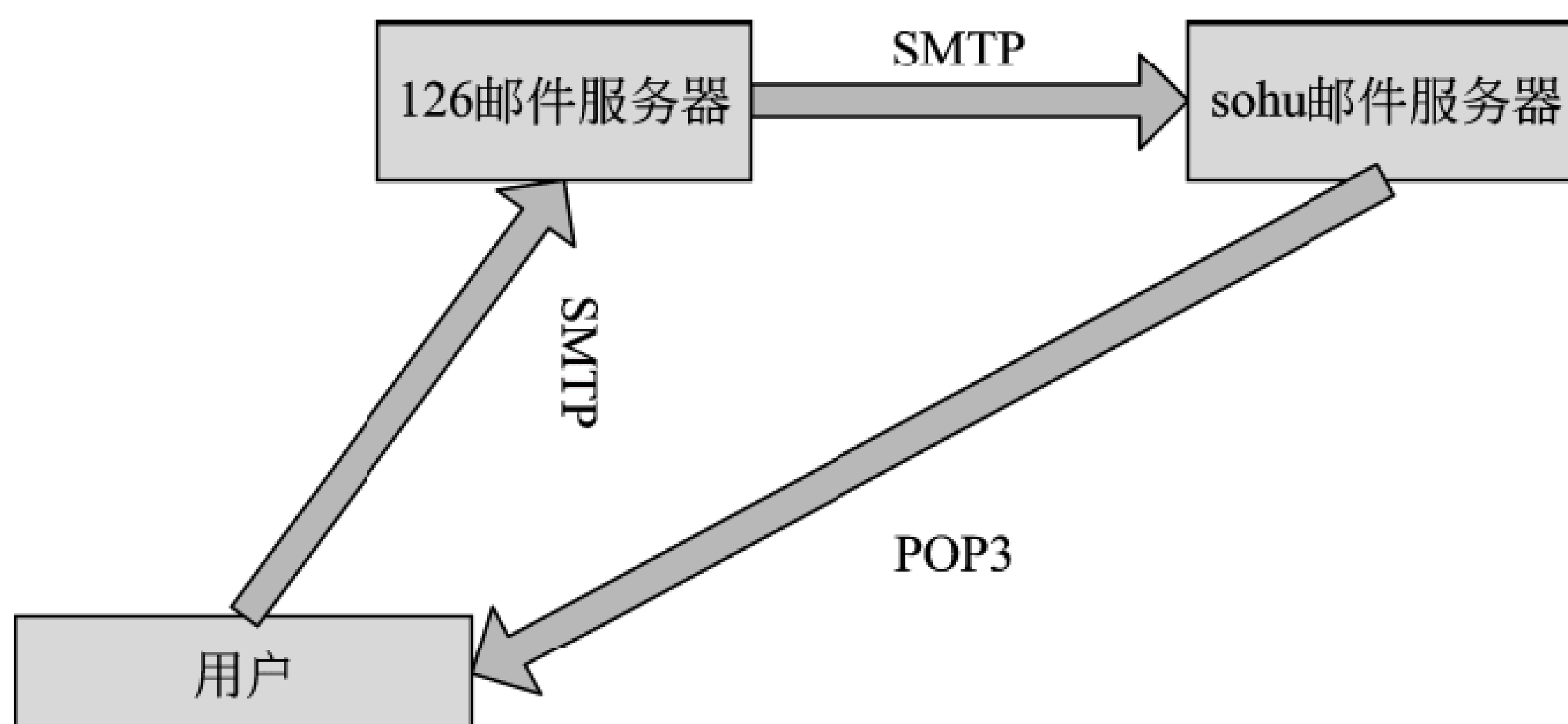


图 11.40　电子邮件的传输过程

1．准备工作

在手工发送和接收电子邮件之前，首先要在 126 邮件服务器中和 sohu 邮件服务器中各申请一个账户：cjgong@126.com 和 cjgong_2@sohu.com。接着查看关于 126 邮件服务器中 SMTP 服务器、POP3 服务器的地址和关于 sohu 邮件服务器中 SMTP 服务器、POP3 服务器的地址。

可以通过网址 http://help.126.com/07/0112/11/34KRNK75007525G1.html 打开 126 网站关于设置 OutLook 软件的页面，从该页面的图片（如图 11.41 所示）中可以发现：

- ❑ SMTP 服务器：smtp.126.com；

- POP3 服务器：pop3.126.com。

可以通过网址 http://help.sohu.com/article_usershow_detail.php?id=960 打开 sohu 网站关于设置 OutLook 软件的页面，从该页面的图片（如图 11.42 所示）中可以发现：

- SMTP 服务器：smtp.sohu.com；
- POP3 服务器：pop3.sohu.com。



图 11.41　设置 OutLook 软件

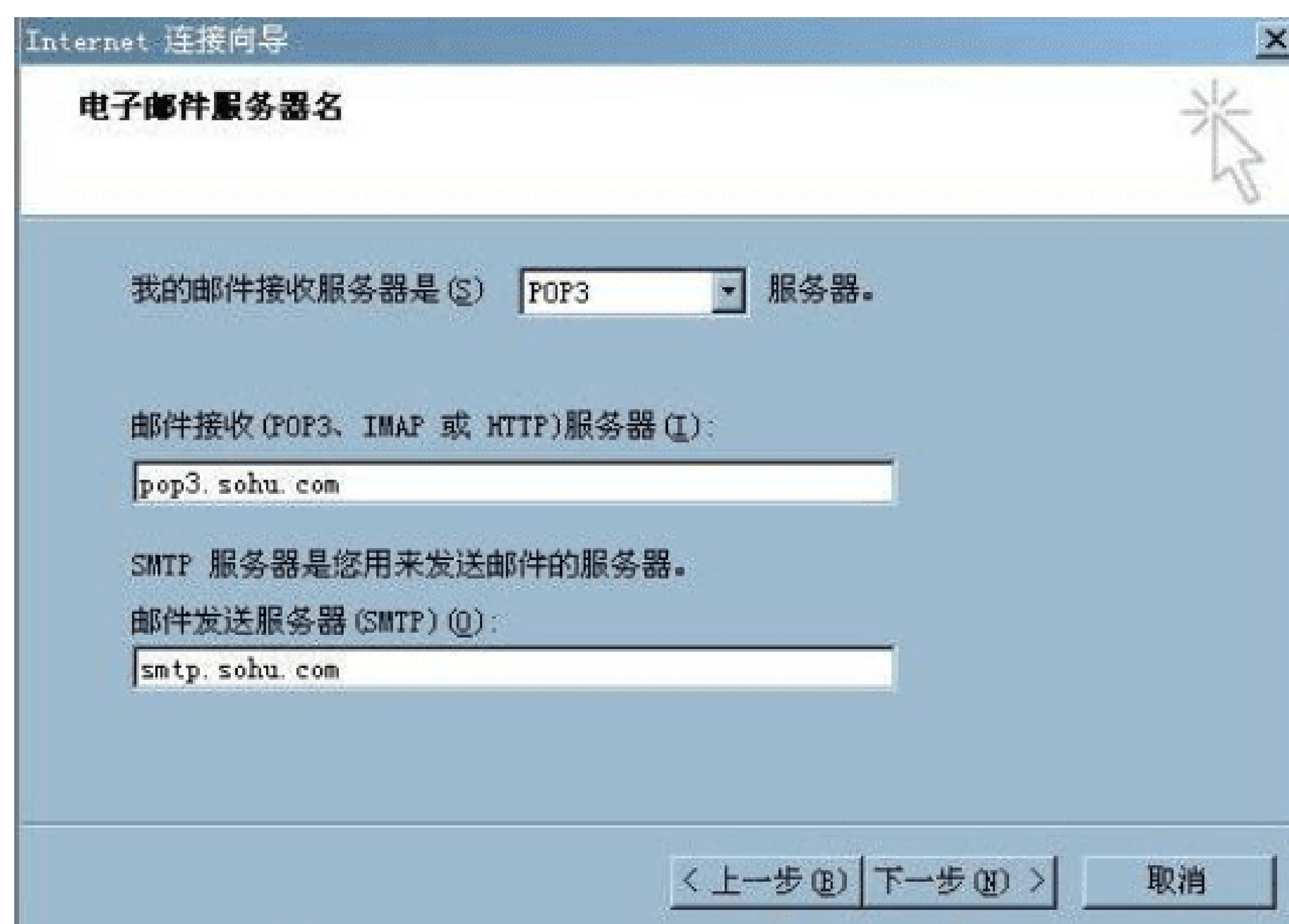


图 11.42　设置 OutLook 软件

2．cjgong向cjgong_2发送一封邮件

首先打开 Dos 命令窗口，利用如下命令来连接关于 126 的 SMTP 服务器（如图 11.43 所示）：

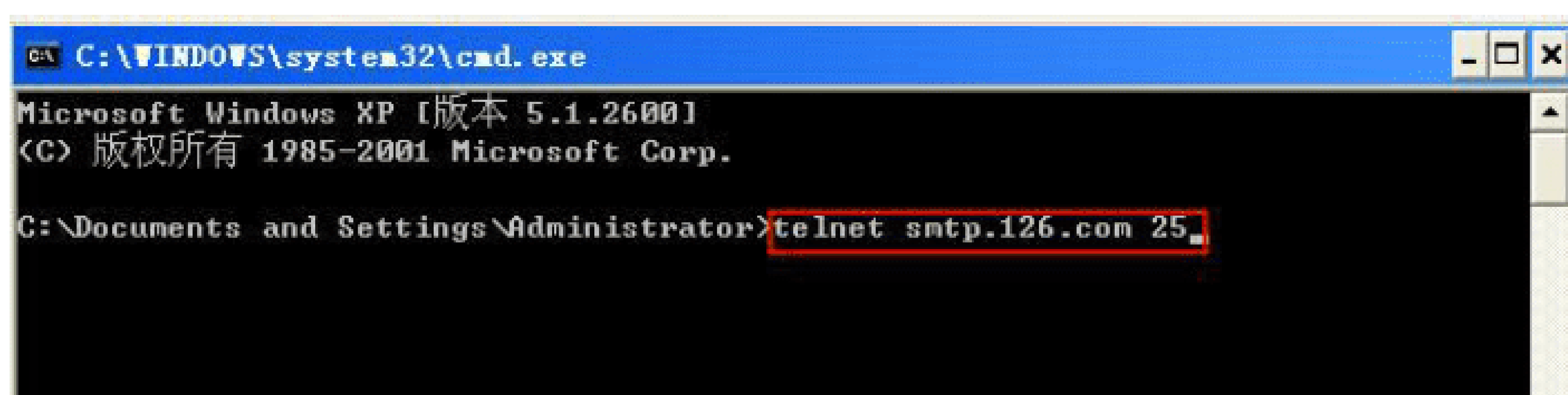


图 11.43　连接 SMTP 服务器

```
telnet smtp.126.com 25
```

其中 telnet 表示连接命令，smtp.126.com 表示服务器的地址，25 表示端口号。当执行完该命令，就会直接进入如图 11.44 所示的命令窗口。

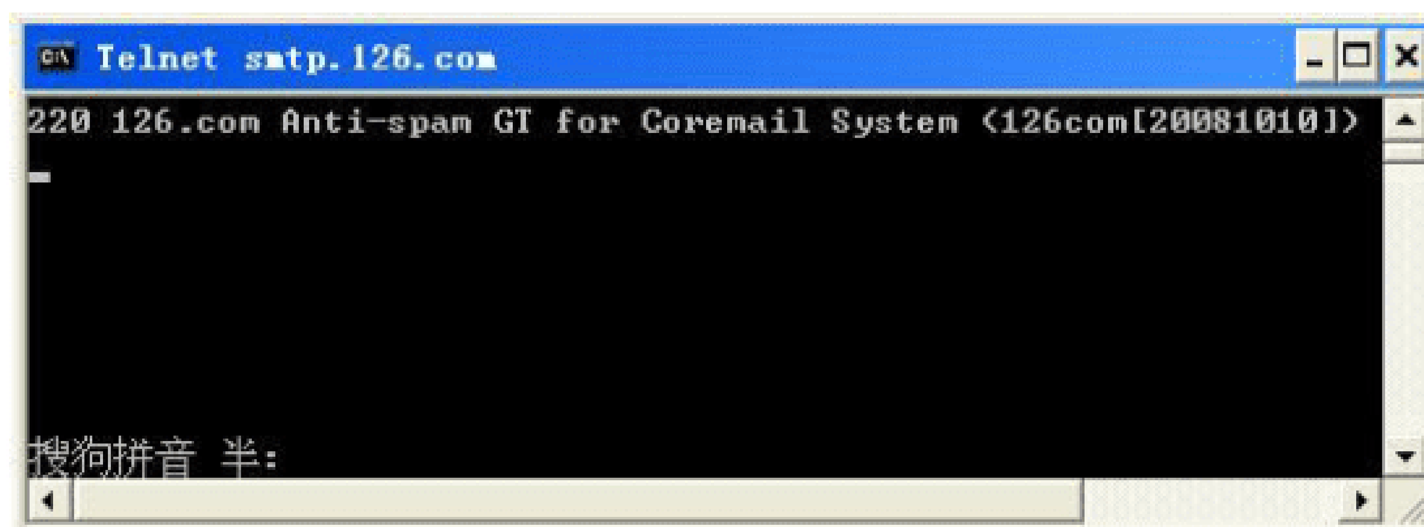


图 11.44　smtp.126.com 命令窗口

接着在图 11.44 所示的窗口中输入如图 11.45 所示的命令。其中各个命令的含义如下。

- 命令 1（ehlo cjgong）：向服务器提交用户的名字，即标示发件人的身份。其中 ehlo

为命令，cjgong 为发件人的身份。

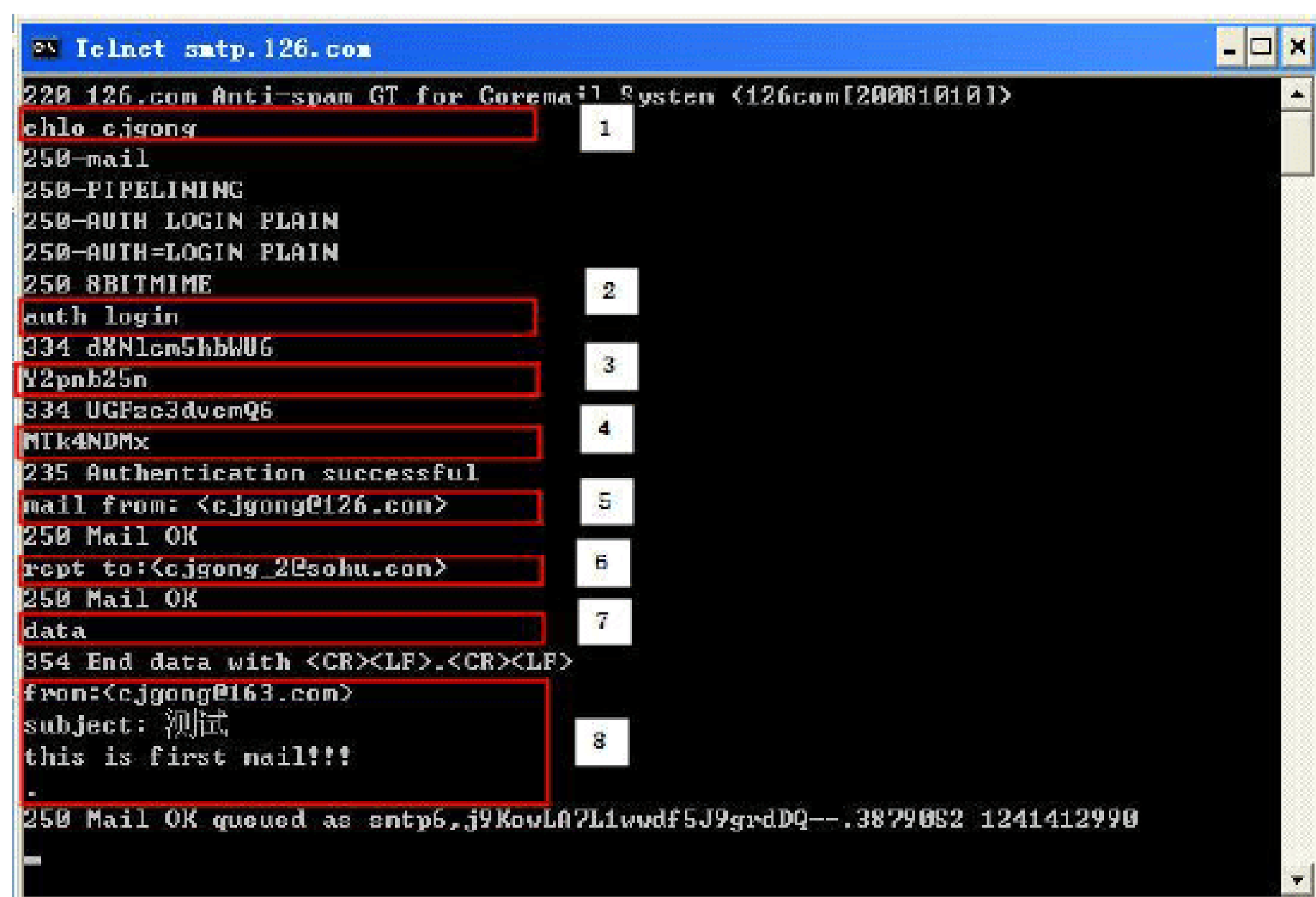


图 11.45　发送邮件

- 命令 2（auth login）：设置认证方式。其中 aut 为命令，而 login 为认证的方式。
- 命令 3（Y2pnb25n）：输入用户名。Y2pnb25n 为用户名 cjgong 的“Base64”的编码。
- 命令 4（MTk4NDMx）：输入密码。MTk4NDMx 为密码 198431 的“Base64”的编码。
- 命令 5（mail from: <cjgong@126.com>）：设置发件人信息，其中 mail from:为命令，而 cjgong@126.com 为发件人的电子邮件账户。
- 命令 6（rcpt to:<cjgong_2@sohu.com>）：设置收件人信息，其中 rcpt to:为命令，而 cjgong_2@sohu.com 为发件人的电子邮件账户。
- 命令 7（data）：表示下面的输入为邮件的数据部分。
- 命令 8（数据部分）：邮件数据部分分为两部分：邮件头和正文。其中邮件头可以设置 3 部分内容：发件人（from:<cjgong@163.com>）、收件人（使用 to:命令，该部分可以省略）和主题（subject:测试）。正文（this is first mail!!!），输入完正文后必须先按下 Enter 键，然后再输入“.”，最后再次按下 Enter 键结束正文的输入。

3. cjgong_2接收邮件

首先打开 Dos 命令窗口，利用如下命令来连接关于 sohu 的 POP3 服务器（如图 11.46 所示）：

```
telnet pop3.sohu.com 110
```

其中 telnet 表示连接命令，pop3.sohu.com 表示服务器的地址，110 表示端口号。当执行完该命令后，就会直接进入如图 11.47 所示的命令窗口。

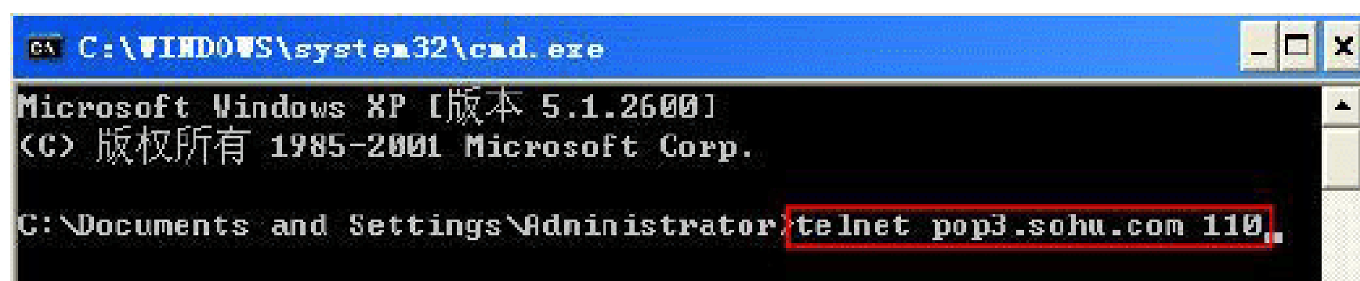


图 11.46　连接 POP3 服务器

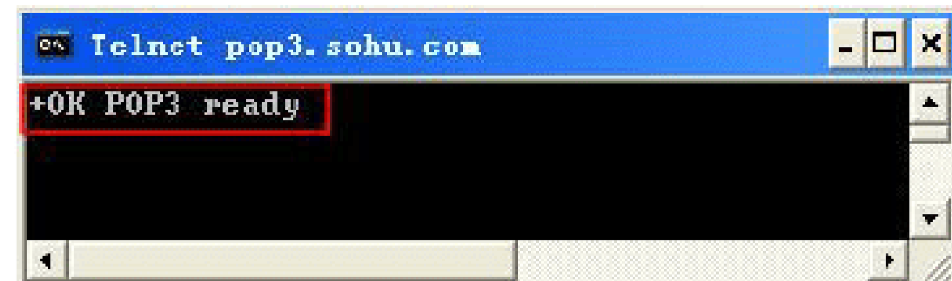


图 11.47　pop3.sohu.com 命令窗口

接着在图 11.47 所示的窗口中输入如图 11.48 所示的命令。其中各个命令的含义如下。

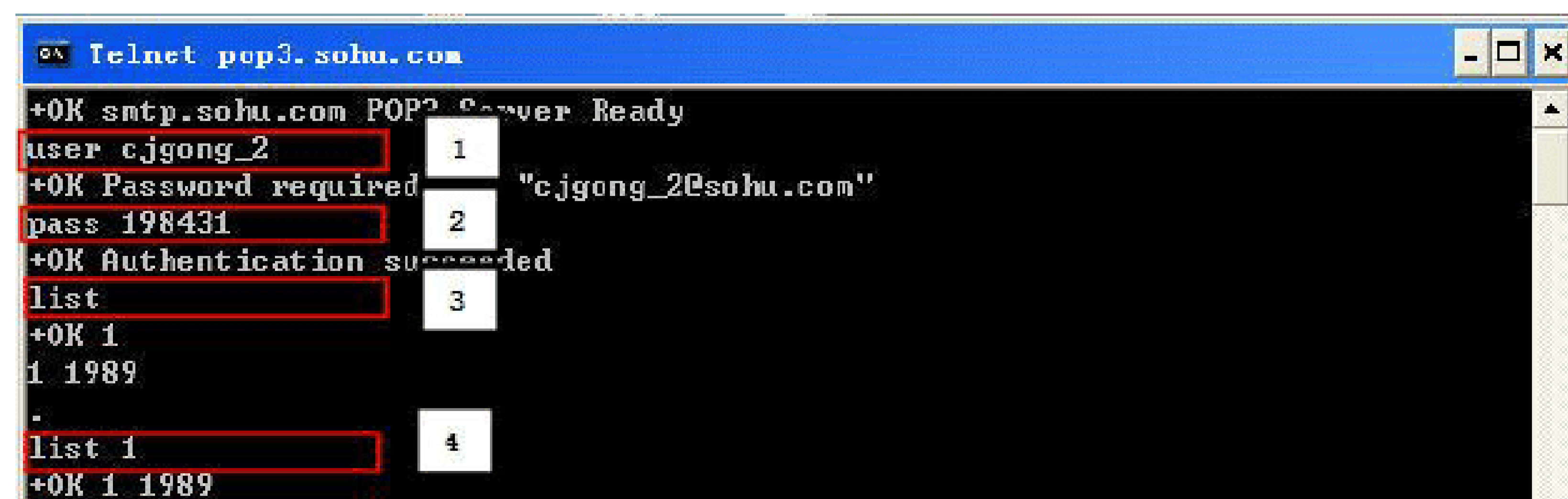


图 11.48　查看邮件

- 命令 1（user cjgong_2）：输入用户名。其中 user 为命令，而 cjgong_2 为用户名。
- 命令 2（pass 198431）：输入密码。其中 pass 为命令，而 198431 为密码。
- 命令 3（list）：显示邮件列表。
- 命令 4（list 1）：显示编号为 1 邮件的信息。

如果想查看特定电子邮件的内容，可以使用 retr 命令。当在 Dos 命令窗口中输入如下命令，就可以出现如图 11.49 所示的窗口。

```
retr 1
```

至此，就完成手工方式的发送和接收邮件。

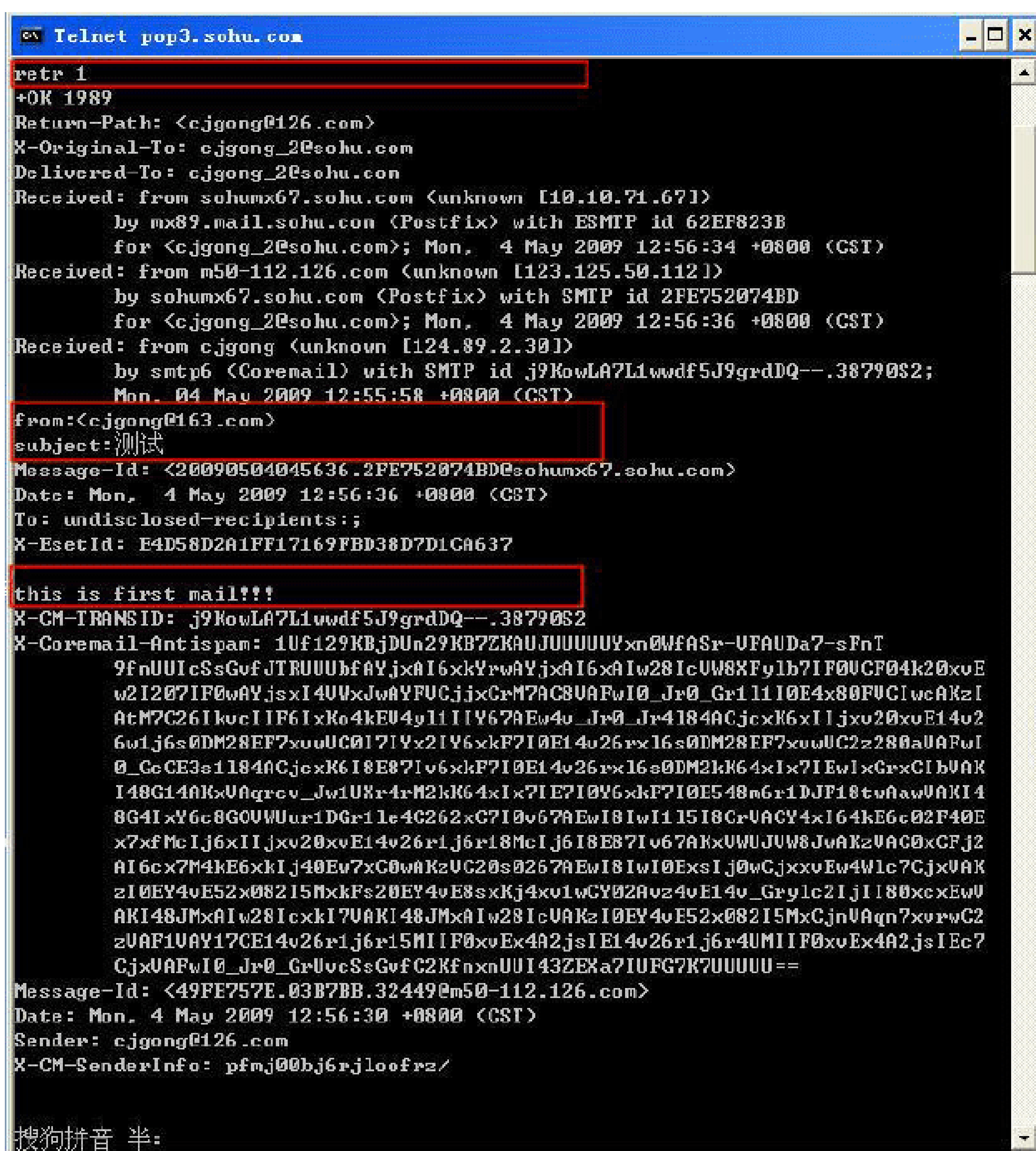


图 11.49　查看邮件内容

11.6.3　设置客户端软件

如果让不熟悉命令的用户在发送和接收电子邮件时都使用手工的方式，那么电子邮件一定不会推广起来。为了让不懂命令的用户也能够很容易地发送和接收电子邮件，许多公司开发了电子邮件客户端，例如微软公司的 OutLook 软件、Foxmail 软件等。本节将讲解如何通过 OutLook 软件发送和接收电子邮件，具体步骤如下。

（1）首先打开 OutLook 软件，通过选择“工具”|“账户”命令打开“Internet 账户”对话框，如图 11.50 所示。

（2）接着单击“新建”按钮，在弹出的菜单中选择“邮件”选项弹出“您的姓名”对话框，如图 11.51 所示。在该对话框的“显示名”文本框中输入“126”。

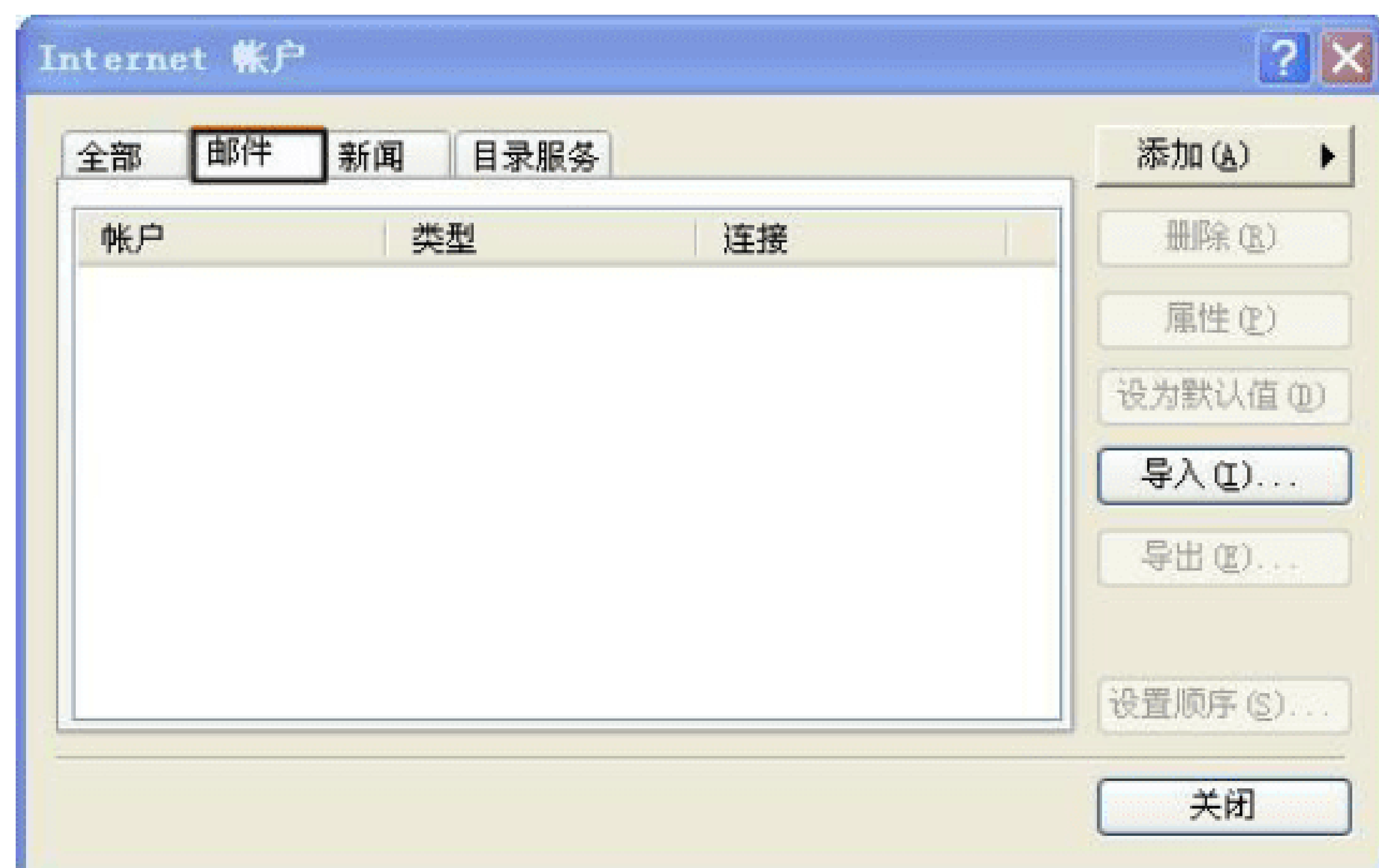


图 11.50　“Internet 账户”对话框



图 11.51　Internet 连接向导

（3）单击“下一步”按钮，进入“Internet 电子邮件地址”对话框（如图 11.52 所示）。在“电子邮件地址”文本框中输入完整的“cjgong@126.com”电子邮件地址。

（4）单击“下一步”按钮，进入“电子邮件服务器名”对话框（如图 11.53 所示）。在该对话框的“接收邮件服务器”文本框中输入“pop3.126.com”，在“发送邮件服务器”文本框中输入“smtp.126.com”。



图 11.52　输入电子邮件地址



图 11.53　电子邮件服务器名

（5）单击“下一步”按钮，进入“Internet Mail 登录”对话框（如图 11.54 所示）。在该对话框的“账户名”文本框中输入“cjgong”，“密码”文本框中输入“198431”。

（6）单击“下一步”按钮，进入“祝贺您”对话框（如图 11.55 所示）。在该对话框中单击“完成”按钮就可以完成对 OutLook 软件的设置。

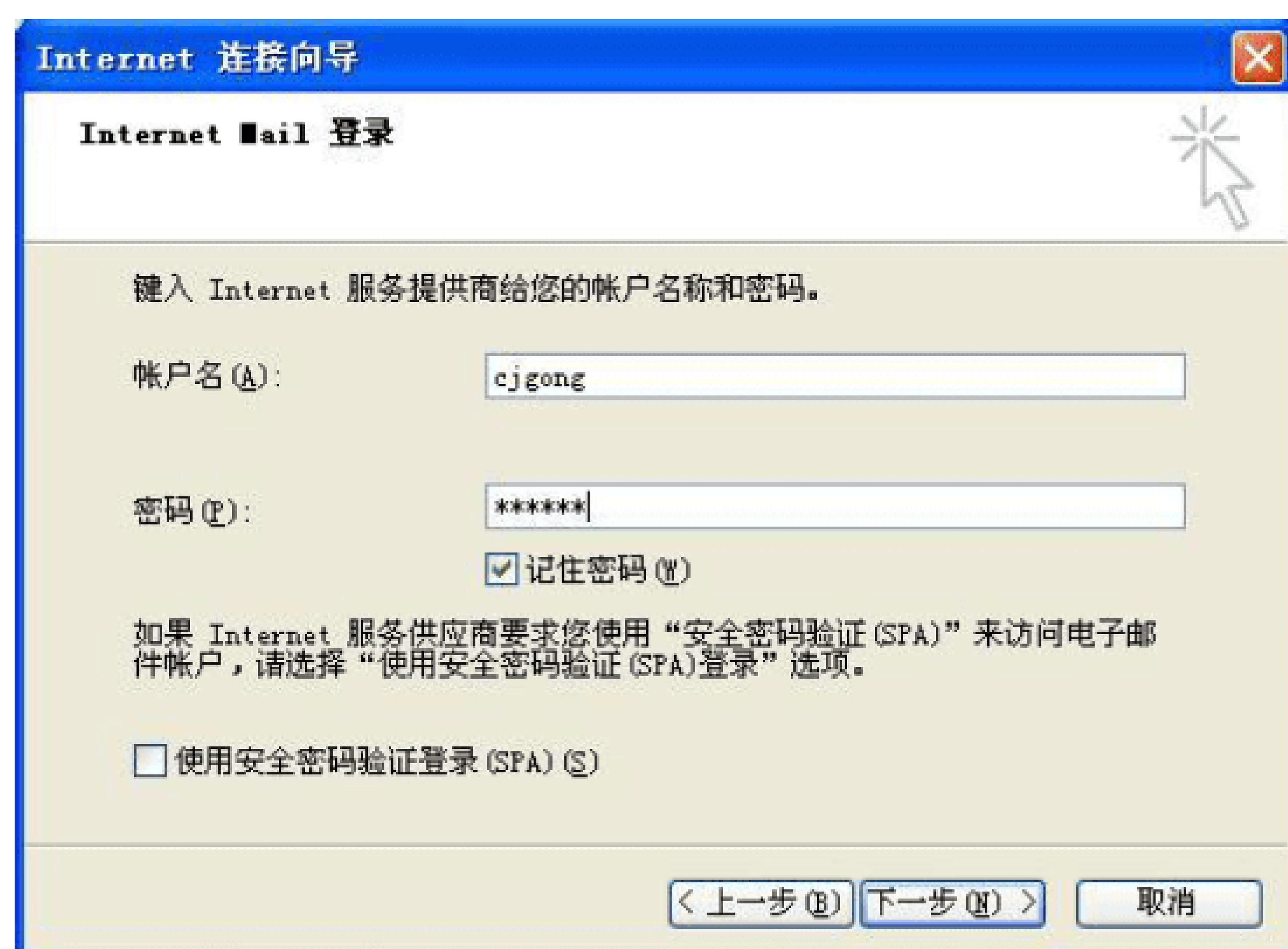


图 11.54　Internet Mail 登录

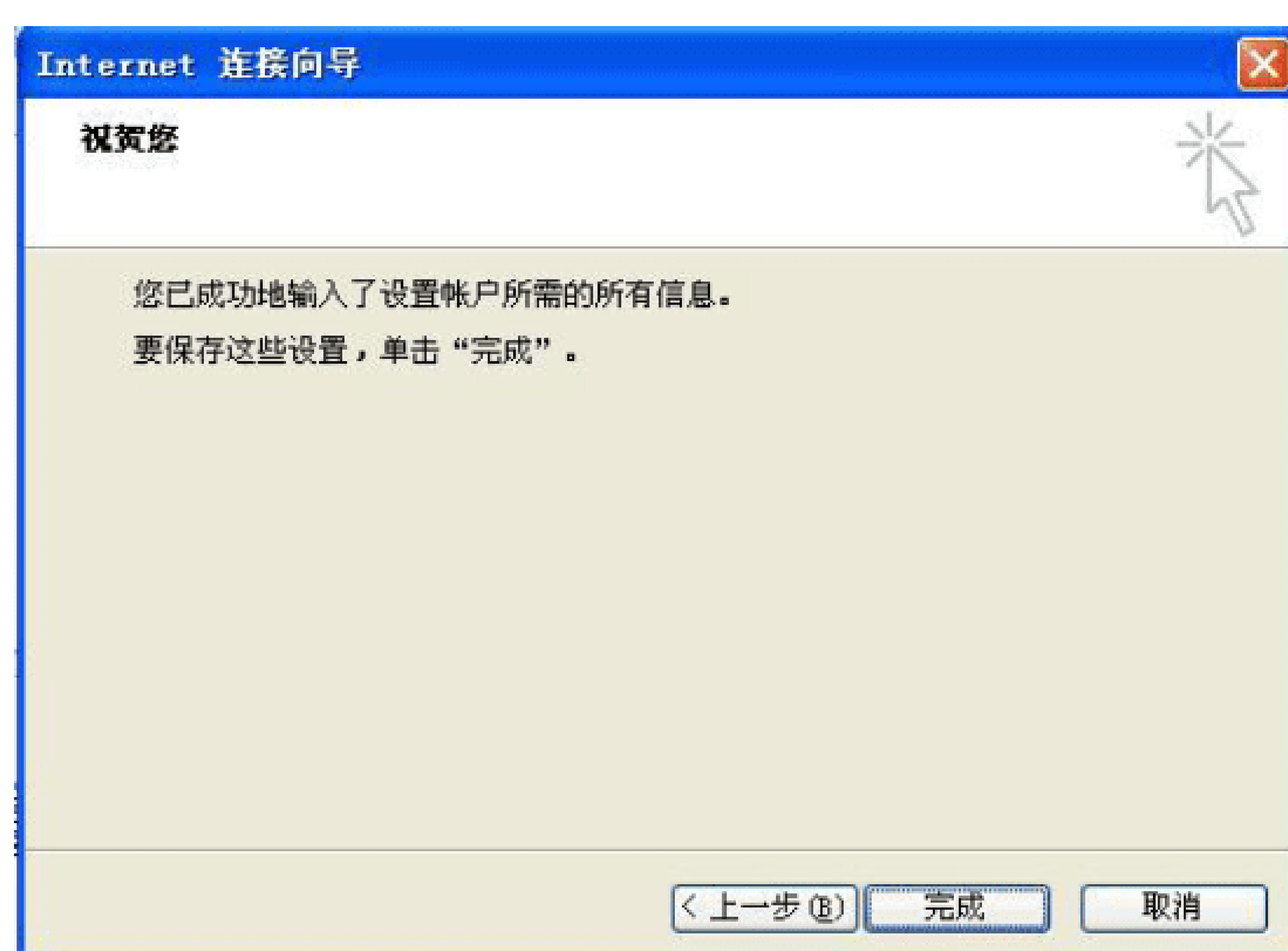


图 11.55　完成 OutLook 软件设置

至此，就完成对 OutLook 软件的设置。这时通过选择“工具”|“发送和接收”|“发送和接收全部邮件”命令就可以下载相应的邮件。

11.6.4　邮件内容的组织结构

通过上面几节的介绍可以清楚知道关于电子邮件发送过程和接收过程，本节将通过 OutLook 软件生成的一个具体电子邮件，来讲解电子邮件内容和该内容需要遵循的格式要求。

1．生成“测试电子邮件.eml”文件

通过 OutLook 软件生成一个名叫“测试电子邮件”的电子邮件，具体步骤如下。

（1）首先打开 OutLook 软件，输入如图 11.56 所示的内容。

（2）在输入具体内容之前，通过如图 11.57 所示的菜单，打开 Outlook Express 对话框。在该对话框中单击“确定”按钮（如图 11.58 所示），把内容设置成纯文本格式。之所以要使用该种格式，因为该种格式的电子邮件最简洁。最后输入“测试语句！！！”内容。

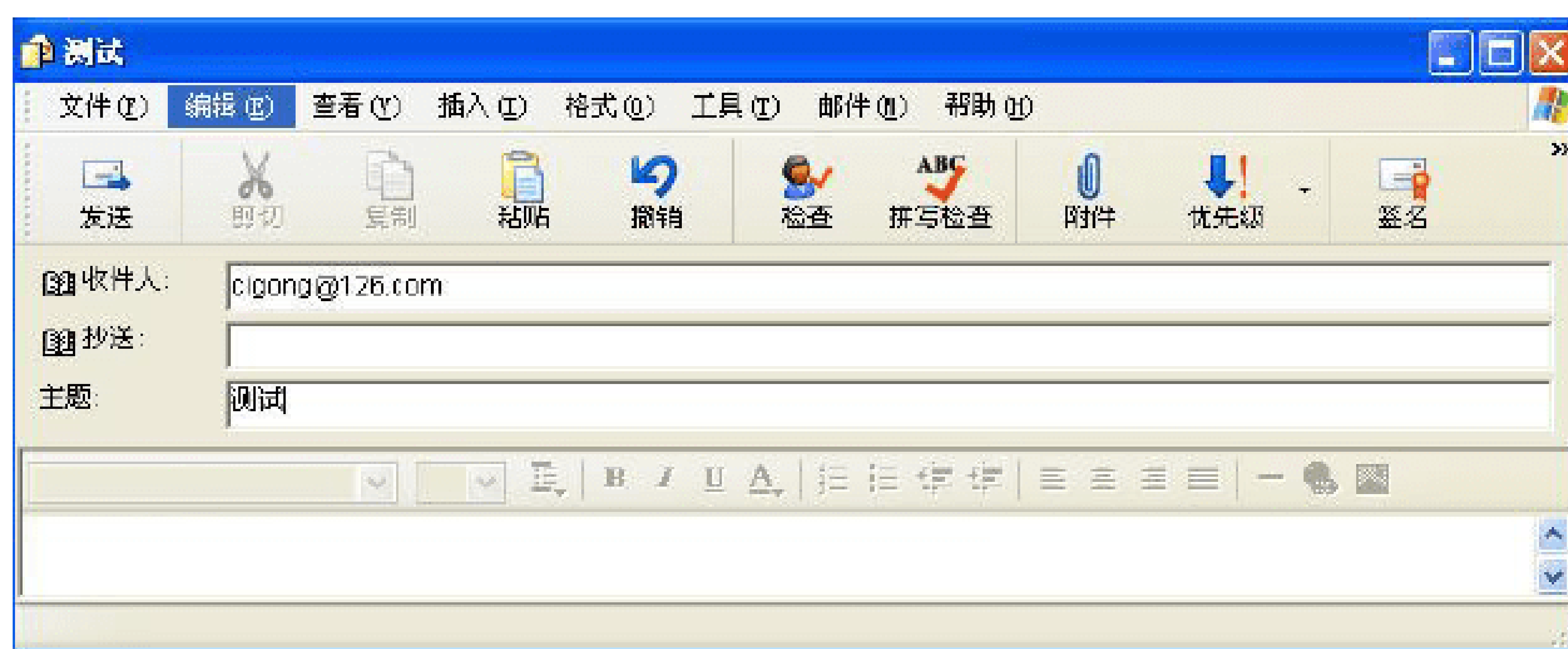


图 11.56　输入内容



图 11.57　菜单

（3）通过选择“文件”|“另存为”命令把该邮件存储为名为“测试电子邮件”的邮件，如图 11.59 所示。



图 11.58　Outlook Express 对话框

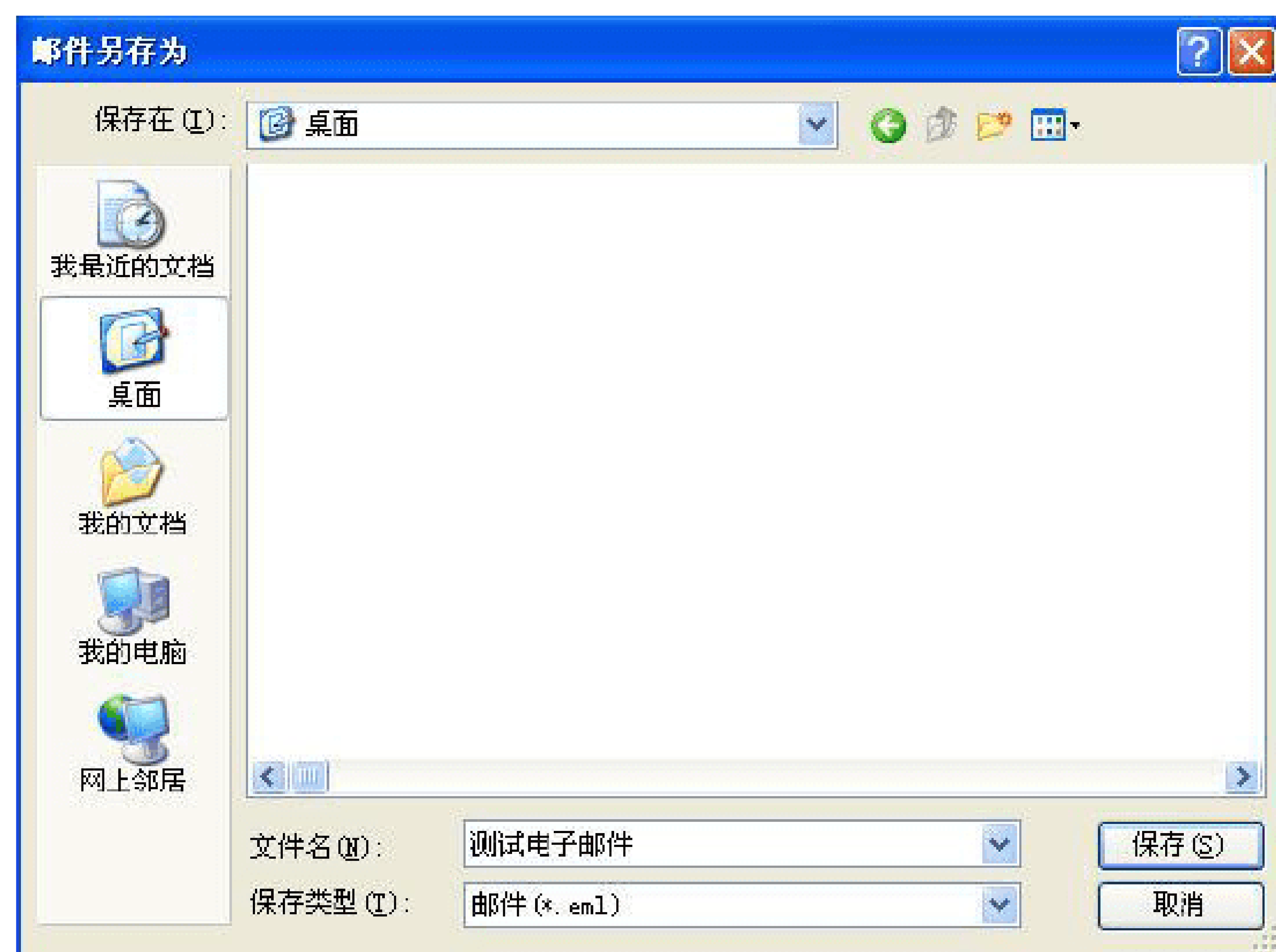


图 11.59　“邮件另存为”对话框

2．测试电子邮件.eml文件的编码格式

电子邮件内容是有格式限制的，通过工具 UltraEdit 打开“测试电子邮件.eml”文件，具体内容如代码 11.7 所示。

代码 11.7　电子邮件内容：测试电子邮件.eml

```
From: "sohu" <cjgong 1@sohu.com>
To: "cjgong@126.com"
Subject: =?gb2312?B?suLK1A==?=
Date: Mon, 11 May 2009 08:12:23 +0800
MIME-Version: 1.0
```

```
Content-Type: text/plain;
    charset="gb2312"
Content-Transfer-Encoding: base64
X-Priority: 3
X-MSMail-Priority: Normal
X-Unsent: 1
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.3350

suLK1NPvvuSjoaOho6E=
```

【代码解析】

根据代码“Content-Transfer-Encoding: base64”可以知道，该邮件的编码格式为 Base64。所谓 Base64：该种编码方式可以实现把二进制数据转换成可打印的 ASCII 字符。该方式的基本原理是将一组连续的字节数据按 6 个 b 进行分组，然后对每组数据用一个 ASCII 字符来表示。由于 6 个 b 能表示 64 个数值，所以可以使用 64 个 ASCII 字符来对应这 64 个数值。64 个 ASCII 字符分别为：

- A—Z；
- a—z；
- +、/；
- 0、1、2、3、4、5、6、7、8、9。

纯文本格式除了可以是 Base64 编码方式外，还可以是 Quoted-printable 编码方式。下面将通过具体步骤来实现把纯文本格式设置为 Quoted-printable 编码方式。

注意：上述代码中，除了最后一段代码为电子邮件的具体内容外，其他代码都是关于电子邮件的格式。

通过选择“工具”|“选项”命令打开如图 11.60 所示的“选项”对话框。在该对话框中首先选择“发送”标签进入“发送”选项卡。然后在“邮件发送格式”选项区域中选择“纯文本”单选按钮，最后单击“纯文本设置”按钮，就可以打开如图 11.61 所示的“纯文本设置”对话框。在该对话框中的邮件格式选项区域中对文本的编码方式选择“括上的可打印项目”选项就可以实现 Quoted-printable 编码方式的设置。



图 11.60　“选项”对话框

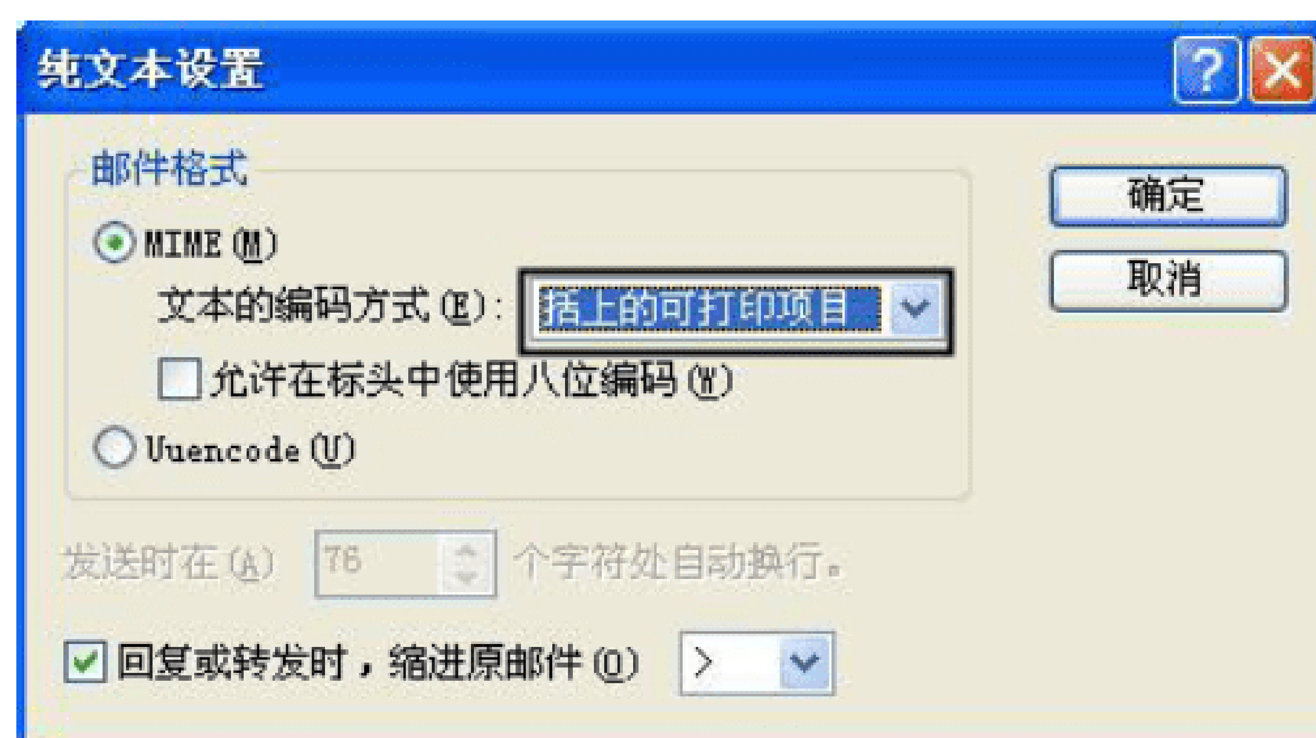


图 11.61　纯文本设置

当把测试电子邮件.eml 文件的内容以 Quoted-printable 编码方式重新编写后，再通过工具 UltraEdit 打开“测试电子邮件 1.eml”文件，具体内容如代码 11.8 所示。

代码 11.8　电子邮件内容：测试电子邮件 1.eml

```
From: "sohu" <cjgong_1@sohu.com>
To: "cjgong@126.com"
Subject: =?gb2312?B?suLK1A==?=
Date: Mon, 11 May 2009 10:33:15 +0800
MIME-Version: 1.0
Content-Type: text/plain;
    charset="gb2312"
Content-Transfer-Encoding: quoted-printable
X-Priority: 3
X-MSMail-Priority: Normal
X-Unsent: 1
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.3350

=B2=E2=CA=D4=D3=EF=BE=E4=A3=A1=A3=A1=A3=A1
```

【代码解析】

- 上述代码中，除了最后一段代码为电子邮件的具体内容外，其他代码都是关于电子邮件的格式。
- 根据代码“Content-Transfer-Encoding: quoted-printable”可以知道，该邮件的编码格式为 quoted-printable。所谓 Quoted-printable：该种编码方式也可以实现把二进制数据转换成可打印的 ASCII 字符。该方式的基本原理是把每个字节数据转换成一个“=”号和该字节的十六进制数据。

注意： Quoted-printable 编码方式只对非 ASCII 字符的数据进行编码转化，而对 ASCII 字符不进行转化。

11.7　小　　结

本章主要介绍了 Java Web 邮件发送系统，该系统基于 JSP+Servlet+JavaBean 解决方案。为了讲清楚邮件发送系统，详细介绍了 3 种形式的邮件发送系统：普通方式电子邮件的发送、HTML 方式电子邮件的发送和携带附件（TXT、DOC）方式电子邮件的发送。

除了利用 JavaMail 组件实现各种邮件发送系统外，本章最后还详细介绍了关于邮件的基本知识。

第 12 章　网络留言板（JSP+Servlet+JavaBean）

拥有一个功能强大且富有个性的网络留言板，是每个大型网站系统的追求。之所以这样，是因为网络留言板是网站与访客之间进行交流的主要手段。一个设计合理、界面优美的网络留言板程序能从侧面体现网站良好的服务，给来访用户留下美好的印象，增强用户对网站的信心。

本章将通过 JSP+Servlet+JavaBean 框架技术来介绍如何实现网络留言板模块，该模块主要包含两种功能：添加留言、浏览留言和管理留言。

12.1　网络留言板原理

网络留言板的功能实际上与现实中留言本的功能一样，不同的只是一个是实体而另一个是虚拟体。当浏览者想留言时，只需要在规定的页面填写相应的内容，然后单击“提交”按钮就可以把留言保存起来。

12.1.1　网络留言板结构框架分析

对于一个大型网站系统来说，实现一个可用的网络留言板功能要考虑的情况十分复杂，例如：如何合理规划数据库、留言内容是否自动删除等。该章将会实现一个比较简单可用的网络留言板，读者可以根据自己的需求自行进行完善。本系统结构框架如图 12.1 所示，项目目录如图 12.2 所示。

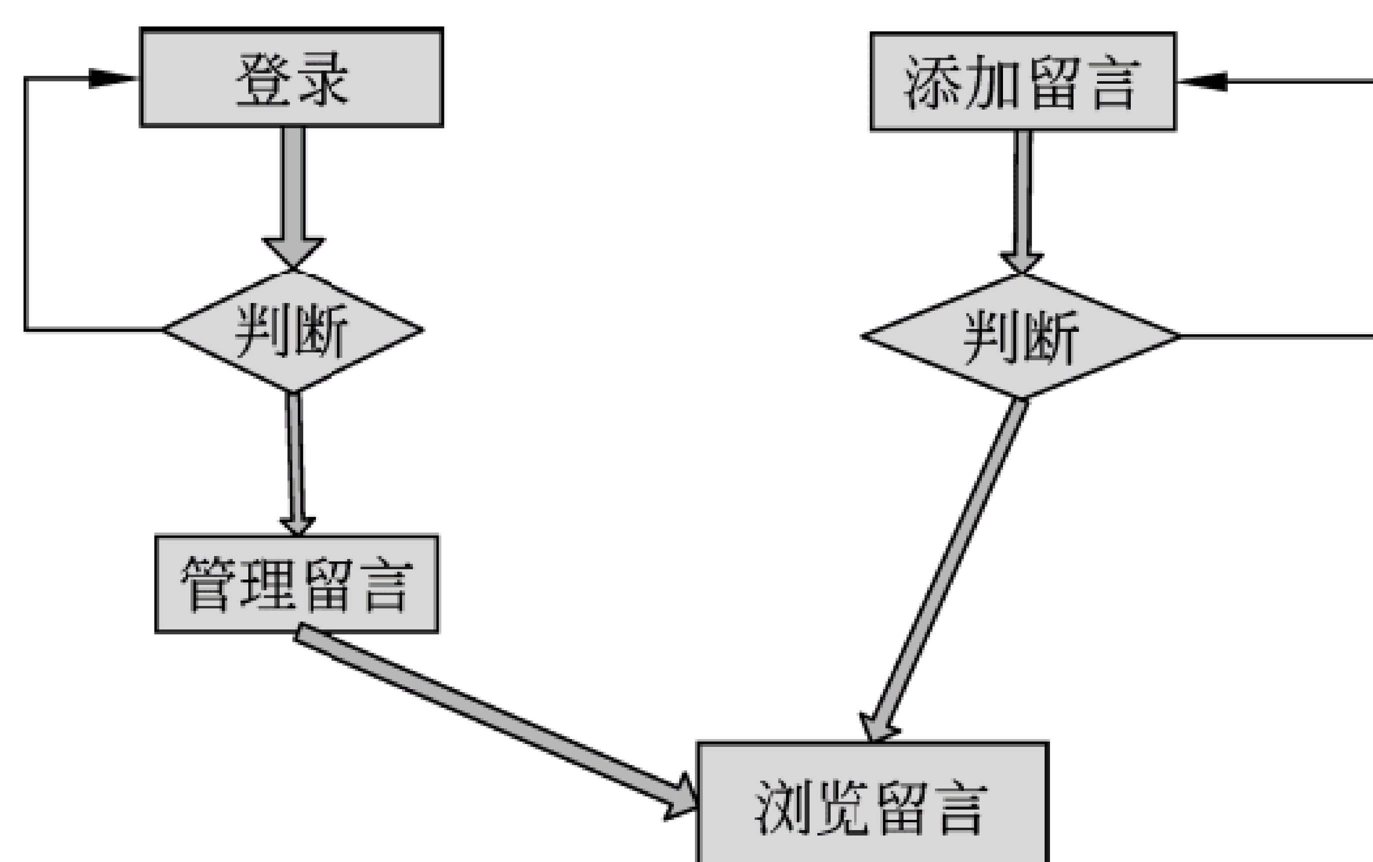


图 12.1　系统流程图

12.1.2　网络留言板功能描述

本节将以直观的方式向读者介绍整个网络留言板模块要实现的功能。这些功能包括添加留言、浏览留言和管理留言。

1．添加留言

浏览者首先通过浏览 addMessage.jsp 网页来添加留言，在该页面（如图 12.3 所示）上填写完相应的内容后，单击“提交”按钮就可以实现添加留言功能。

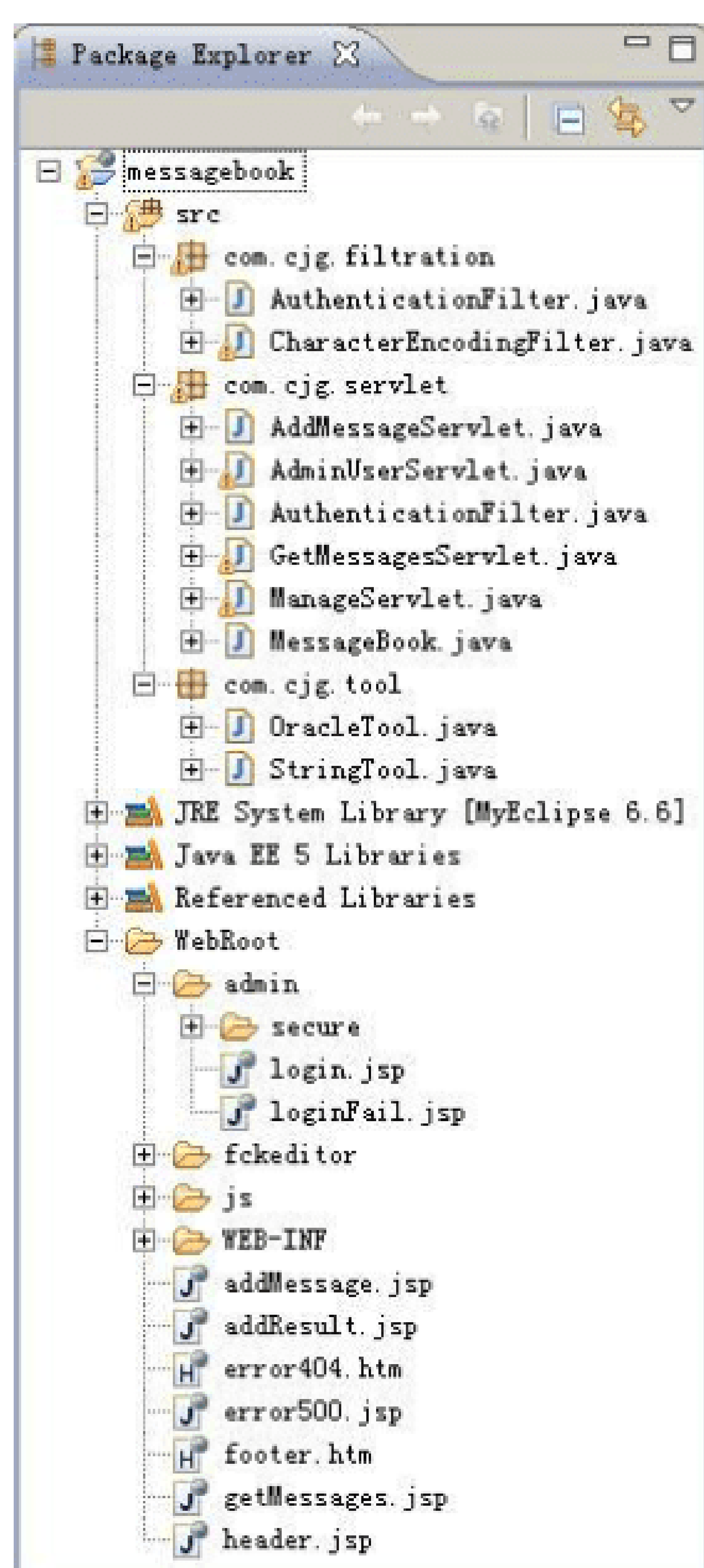


图 12.2　项目目录



图 12.3　添加留言

当浏览者完成添加留言后，这时就会转到显示添加留言结果的页面：当添加留言成功后如图 12.4 所示，当添加留言失败后如图 12.5 所示。如果浏览者还想接着添加留言，可以单击“添加新的留言”链接。



图 12.4　添加留言成功结果



图 12.5　添加留言失败结果

2. 浏览留言

如果浏览者想在添加完留言后浏览所有的留言内容，可以单击图 12.4 中的“查看所有留言的内容”链接，这样就会转到显示留言内容的 getMessages.jsp 网页（如图 12.6 所示）。

3. 管理留言

只有登录成功的浏览者才能对留言进行删除、修改等操作。当浏览者浏览完所有的留言内容后，单击“管理员登录”链接就会进入登录页面。浏览者会通过如图 12.7 所示的页面进行登录，如果登录失败就会转到如图 12.8 所示的失败页面。在该



图 12.6　浏览留言

页面中如果想重新登录，可以单击“重新登录”链接转到登录页面。



图 12.7　登录页面

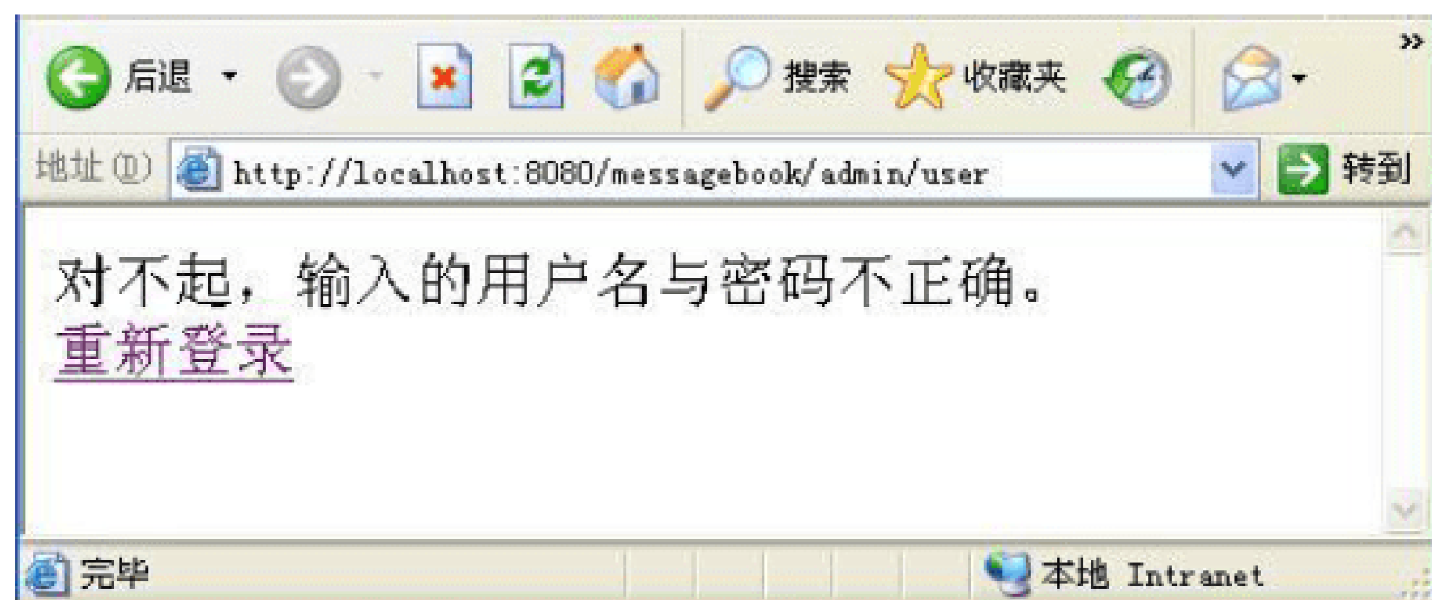


图 12.8　登录失败

登录成功后就会进入管理留言页面（如图 12.9 所示），在该图中可以通过单击“编辑”、“删除”链接来实现对留言内容的修改和删除。

4．编辑留言

在管理留言页面中，单击“编辑”链接就会进入修改留言页面（如图 12.10 所示），在该页面中修改相应的内容后，单击“提交”按钮，就会进入编辑留言成功页面（如图 12.11 所示）。



图 12.9　管理页面



图 12.10　修改留言

5．删除留言

如果想删除留言内容，只需要在图 12.9 所示的管理留言页面中单击“删除”链接就可以转到如图 12.12 所示的删除成功页面。

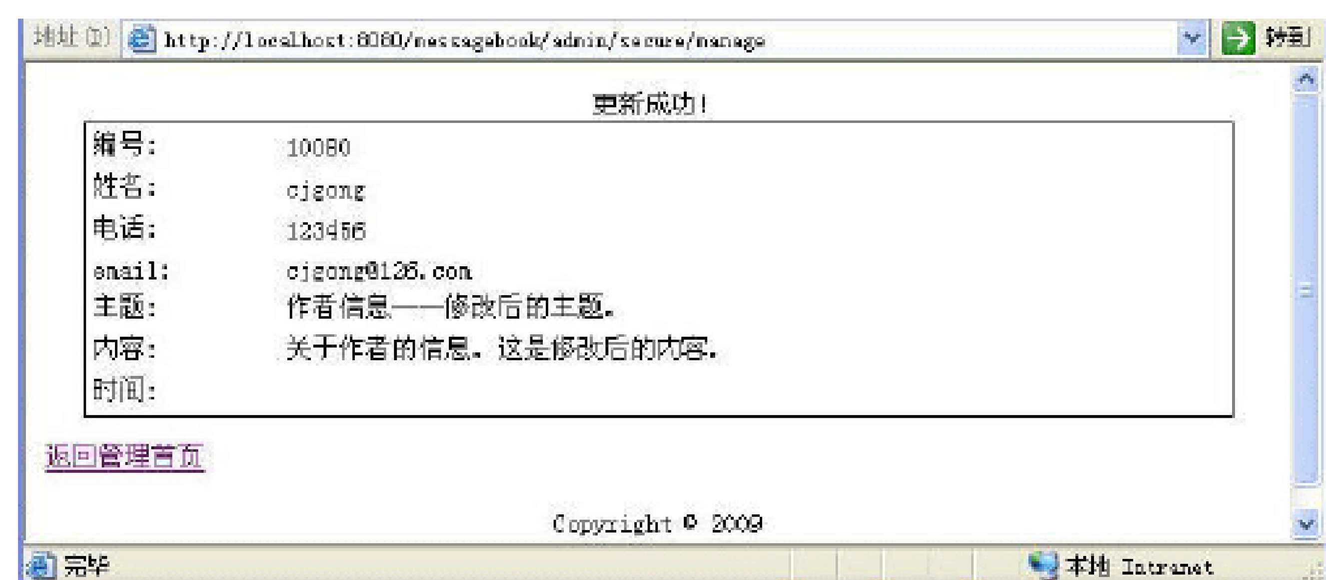


图 12.11　修改留言成功



图 12.12　删除成功

12.2　添 加 留 言

本节通过 JSP+Servlet+JavaBean 框架技术来实现添加留言功能，在该功能中包含 2 个 JSP 页面、1 个 JavaBean 和 2 个 Servlet 程序：addMessage.jsp、addResult.jsp、MessageBook.java、AddMessageServlet.java 和 OracleTool.java（OracleTool 数据库工具类）。这些程序之间的关系如图 12.13 所示。

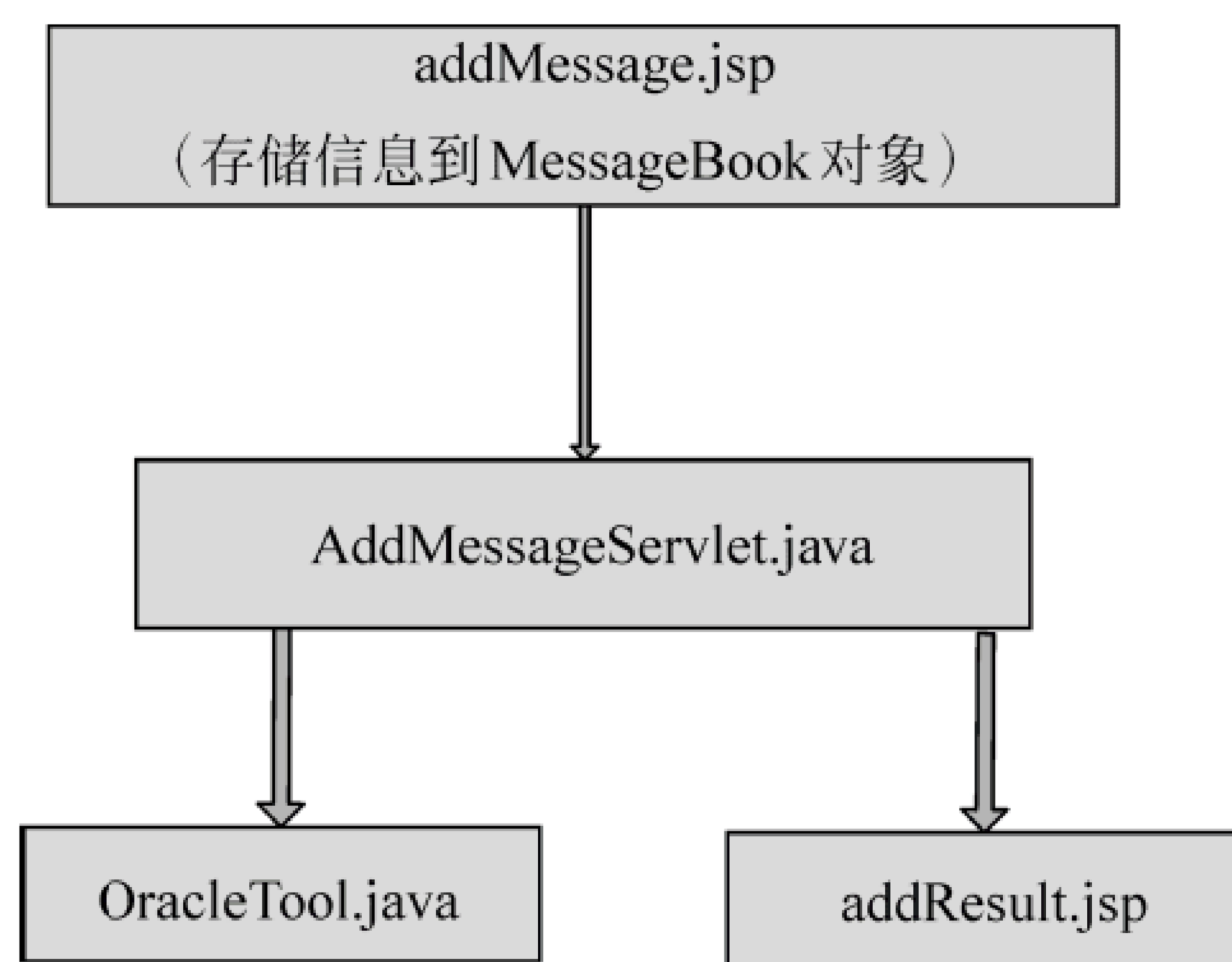


图 12.13　程序关系图

（1）浏览者通过浏览器打开 addMessage.jsp 页面，然后填写留言的内容。当单击“提交”按钮后，就会把留言的内容存储到 MessageBook 组件里，并传递该对像于 Add-Message-Servlet 程序。

（2）当 AddMessageServlet 程序接受到传递过来的留言内容时，会检验这些留言内容。当留言内容符合要求后，就会生成 OracleTool 对象，最后调用该对象的 update()方法更新数据库。

（3）最后，AddMessageServlet 程序会传递一个名叫 message 的变量给 addResult.jsp 页面，addResult 程序会根据变量 message 的值显示添加留言的结果。

12.2.1　添加留言页面

addMessage.jsp 页面用来让浏览者填写留言，在该页面中使用 FCKeditor 编辑器。浏览者需要在姓名、E-mail、电话、主题文本框和内容 FCKeditor 编辑器中填写相对应的内容，才能完成留言的填写，具体内容如代码 12.1 所示。

代码 12.1　添加留言页面：addMessage.jsp

```
...
<style>                                            <!--样式表-->
* {
    font-family: "宋体";
    font-size: 14px
}
</style>
<!--引入外部 javascript 文件-->
<script
type="text/javascript"src="<%=context%>/js/validation-framework.js"></s
cript>
<script                                                   type="text/javascript"
src="<%=context%>/fckeditor/fckeditor.js"></script>
<p align="center">
    请您输入留言
</p>
<p align="center">                                 <!--查看留言的链接-->
    <a href="<%=context%>/servlet/getMessages">查看留言</a>
</p>
```

```
<!--表单内容，传递给/servlet/addMessage -->
<form id="form1" name="form1" method="post"
    action="<%=context%>/servlet/addMessage"onsubmit="return  doValidate
    (this)">
    <table width="650" height="400" border="0" align="center">
        <tr>
            <td width="150">
                姓名：
            </td>
            <td width="500">
                <input name="name" type="text" id="name" size="40"
                maxlength="20" />
            </td>
...
        <tr>
            <td valign="top">
                内容：
            </td><td>
                <script type="text/javascript"><!--使用 FCKeditor 编辑器-->
                var oFCKeditor = new FCKeditor("content");
                oFCKeditor.BasePath = '<%=context%>/fckeditor/' ;
                oFCKeditor.Height  = 300 ;
                oFCKeditor.ToolbarSet = 'Basic';
                oFCKeditor.Create() ;
                </script>
            </td></tr><tr><td></td><td>
                <input type="submit" name="Submit" value="提交" />
                <input type="reset" name="Reset" value="重置" />
            </td></tr>
    </table>
</form>
...
```

【代码解析】

上述代码只是一个简单的表单页面，在该页面中利用了 FCKeditor 在线编辑器。从上述代码中可以发现在查看留言和添加留言时，都是由映射地址为/servlet/addMessage 的 Servlet 程序来处理。

12.2.2　处理留言的内容

要实现对留言内容的处理首先需要获取该对象，所以需要一个针对留言内容的虚拟对象，该对象最好能对应到一个 JavaBean 对象上，所以在本项目中编写了 MessageBook.java 类，具体内容如代码 12.2 所示。

代码 12.2　留言内容对象：MessageBook.java

```
...
public class MessageBook {
    private String content;                          //定义了内容属性
    private String email;                            //定义了 E-mail 属性
    private Integer id;                              //定义了 ID 属性
    private String name;                             //定义了名字属性
    private String phone;                            //定义了电话属性
    private String time;                             //定义了时间属性
```

```
    private String title;                                //定义了主题属性
    //省略属性 content、email、id、name、phone、time 和 title 的配置
...
}
```

AddMessageServlet.java 是服务器端的 Servlet 程序，用来实现对 addMessage.jsp 页面中传送过来的信息进行处理。代码 12.3 用来实现对信息的保存。

代码 12.3　处理留言：AddMessageServlet.java

```
...
public class AddMessageServlet extends HttpServlet {
    //编写 doPost()方法
    public  void  doPost(HttpServletRequest  request,  HttpServletResponse
response) throws ServletException, IOException {
        //定义插入的 SQL 语句
        String sql = "insert into MessageBook (id,name,email,phone,title,
        content,time) values(gb seq.nextval,?,?,?,?,?,?)";
        int result = 0;                         //定义一个表示结果的变量
        String message = "";                    //定义一个添加信息结果变量
        request.setCharacterEncoding("utf-8");
        String name = request.getParameter("name");
                                                //接受浏览者填写的 name 参数
        String title = request.getParameter("title");
                                                //接受浏览者填写的 title 参数
        if (StringUtil.validateNull(name)) {    //判断用户名不为空
            message = "对不起，姓名不能为空，请您重新输入！<br>";
        } else if (StringUtil.validateNull(title)) {   //判断主题不为空
            message = "对不起，主题不能为空，请您重新输入！<br>";
        } else {
            //转换时间格式
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd hh:mm:
            ss");
            //形成参数数组
            String param[] = { StringUtil.filterHtml(name), StringUtil.
            filterHtml(request.getParameter("email")),
                    StringUtil.filterHtml(request.getParameter("phone")),
                    StringUtil.filterHtml(title),
                    request.getParameter("content"), sdf.format(new java.
                    util.Date()) };
            //生成 OracleTool 对象来操作数据库
            OracleTool db = new OracleTool("java:/comp/env/jdbc/
            oracleds");
            db.init();                              //调用 init()方法
            result = db.update(sql, param);         //更新数据库，返回结果
            if (result == 0) {
                message = "对不起，添加留言不成功，请您重新输入！<BR>";
            } else {
                message = "祝贺您，成功添加留言。<BR>";
            }
        }
        //把注册信息保存到属性中
        request.setAttribute("message", message);
        //转向到 addResult.jsp 页面，并显示注册信息
        request.getRequestDispatcher("/addResult.jsp").forward(request,
            response);
    }
```

```
}
```

接着在 web.xml 文件里实现对 Servlet 类进行配置。

```
<!--配置 AddMessageServlet 类路径-->
<servlet>
    <servlet-name>AddMessageServlet</servlet-name>
    <servlet-class>com.cjg.servlet.AddMessageServlet</servlet-class>
</servlet>
<!--配置 AddMessageServlet 映射路径-->
<servlet-mapping>
    <servlet-name>AddMessageServlet</servlet-name>
    <url-pattern>/servlet/addMessage</url-pattern>
</servlet-mapping>
```

【代码解析】

- ❑ 执行 AddMessageServlet 程序时，首先通过 request.getParameter()方法获取 addMessage.jsp 页面传递过来的 name 和 title 参数。
- ❑ 获取 name 和 title 参数后，就会通过工具类判断是否为空，当两个参数有任何一个为空时，就会输出错误信息。然后把传递过来的所有参数通过工具类组成参数数组。
- ❑ 通过数据库工具类把参数数组中的数据存储到数据库中，然后获取数据库操作后的结果变量 result。最后根据变量 result 的值来设置结果变量 message 的值。
- ❑ 把结果变量 message 的值存储到属性中，然后转向到 addResult.jsp 页面中显示变量 message 的值。

添加完留言后，就会转到显示留言添加结果的 addResult.jsp 页面，代码 12.4 实现了对留言添加结果的显示。

代码 12.4　显示留言添加结果：addResult.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
...
<style>
* { font-family: "宋体";font-size: 14px}
</style>
<%=request.getAttribute("message")%>            <!--输出属性message 的值-->
<a href="<%=context %>/addMessage.jsp">添加新的留言</a><BR>
<a href="<%=context %>/servlet/getMessages">查看所有留言内容</a><BR>
...
```

12.3　浏　览　留　言

本节通过 JSP+Servlet+JavaBean 框架技术来实现浏览留言功能，该功能包含 JSP 页面、JavaBean 和 Servlet 程序：getMessages.jsp、MessageBook.java、GetMessagesServlet.java 和 OracleTool.java（OracleTool 数据库工具类）。这些程序之间的关系如图 12.14 所示。

（1）首先浏览者会直接调用 GetMessagesServlet 程序，接着在该程序中会生成 OracleTool 对象，最后调用该对象的 query()方法查找所有的留言内容。

（2）GetMessagesServlet 程序会把查找到的所有留言内容保存到 list 对象中，然后把该对象转向到 getMessages.jsp 页面，最后在该页面中显示出 list 对象。

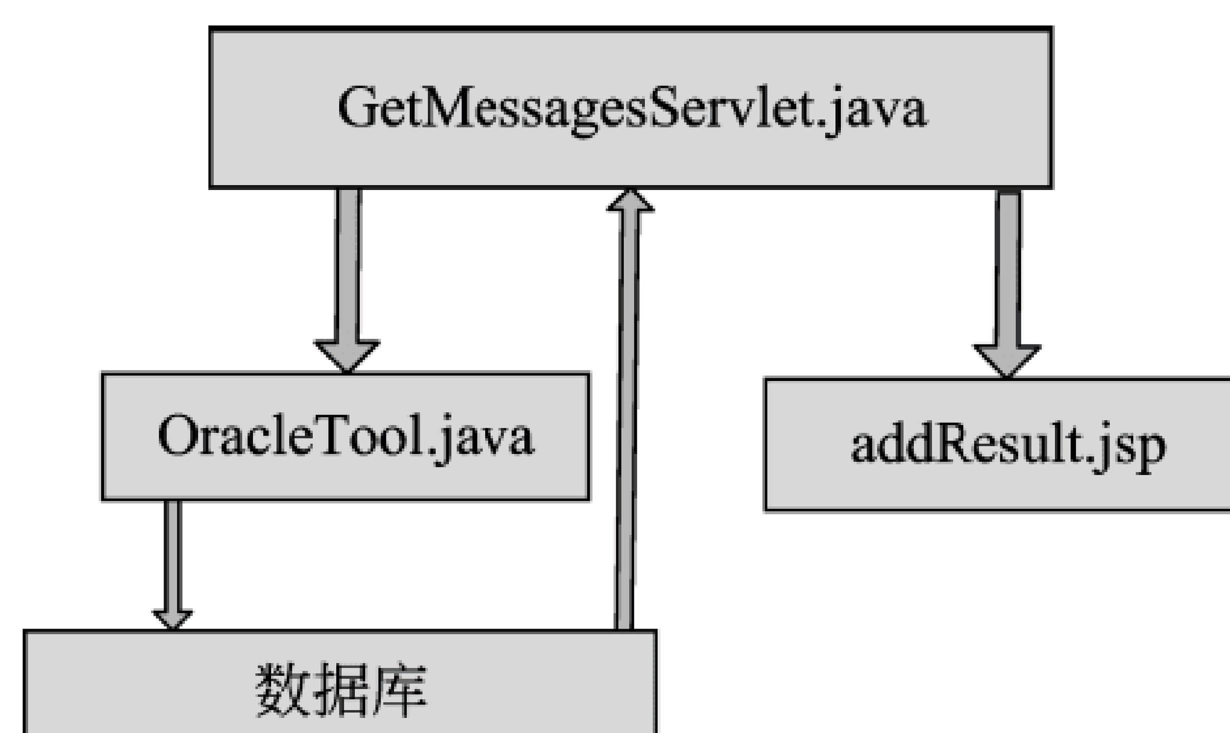


图 12.14　程序关系图

12.3.1　获取所有留言内容

GetMessagesServlet.java 是服务器端的 Servlet 程序，用来从数据库中获取所有的留言内容，然后把这些内容显示在相应的页面中。代码 12.5 用来实现留言内容的获取。

代码 12.5　获取留言页面：GetMessage.jsp

```
...
public class GetMessagesServlet extends HttpServlet {
    //编写 doGet()方法
    public  void  doGet(HttpServletRequest  request,  HttpServletResponse
    response) throws ServletException, IOException {
        String sql = "select * from MessageBook order by id desc"
                                                    //定义一个 SQL 变量
        List list = null;                           //定义一个 list 变量
        OracleTool db  =  new  OracleTool("java:/comp/env/jdbc/oracleds");
                                                    //创建一个工具类
        db.init();                                  //初始化工具类
        //返回获取到结果
        list = (List) db.query(sql, null, new BeanListHandler(MessageBook.
        class));
        HttpSession session = request.getSession();
        session.setAttribute("results", list);      //把获取结果存储到属性
        //转向到 getMessages.jsp 页面
        response.sendRedirect("/getMessages.jsp");
    }
}
```

接着在 web.xml 文件中对该 Servlet 类进行配置。

```
<!--配置 GetMessagesServlet 类路径-->
<servlet>
  <servlet-name>GetMessagesServlet</servlet-name>
  <servlet-class>com.cjg.servlet.GetMessagesServlet</servlet-class>
</servlet>
<!--配置 GetMessagesServlet 映射路径 -->
<servlet-mapping>
  <servlet-name>GetMessagesServlet</servlet-name>
  <url-pattern>/servlet/getMessages</url-pattern>
</servlet-mapping>
```

【代码解析】

- ❑ 首先通过数据库工具类获取所有的留言内容。在使用工具类的 query()方法时，第 1 个参数为查询数据的 SQL 语句变量。第 2 个参数为占为符，即当没有参数时用 NULL 字符串来代替。第 3 个参数为了达到一条记录就是一个 MessageBook.对象的效果，用 BeanListHandler 对象来代替。

- 把从数据库中获取的结果 list，先存储到 Session 对象中，然后转向到 getMessages.jsp 页面显示出来。

12.3.2　浏览留言页面

getMessages.jsp 页面用来显示所有的留言内容，当浏览者单击“查看所有留言的内容”链接后就会转到该页面，该页面会显示通过 GetMessagesServlet 类获取的留言内容。代码 12.6 实现了浏览留言的页面。

代码 12.6　浏览留言页面：getMessages.jsp

```
...
<center>
    <!--添加留言内容的超级链接-->
    <a href="<%=context%>/addMessage.jsp">添加新的留言内容</a><br>
    留言内容<br><br>
    <%
        List list = (List)session.getAttribute("results");
        //从 Session 对象中获取名为 results 属性
        //遍历 list，然后把获取的留言内容显示出来
        if(list!=null){
        for (int i = 0; i < list.size(); i++) {
            MessageBook gb = (MessageBook) list.get(i);
    %>
    <table width="600" border="1" bordercolor="000000"
        style="table-layout: fixed; word-break: break-all">
        <tr>
            <td width="100" bordercolor="ffffff">
                编号:</td>
            <td width="500" bordercolor="ffffff"><%=gb.getId()%></td>
        </tr><tr>
            <td bordercolor="ffffff">
                姓名:</td>
            <td bordercolor="ffffff"><%=gb.getName()%></td>
        </tr><tr>
            <td bordercolor="ffffff">
                电话:</td>
            <td bordercolor="ffffff"><%=StringUtil.chanageNull(gb.
            getPhone(), "没填")%></td>
        </tr><tr>
            <td bordercolor="ffffff">
                email:</td>
            <td bordercolor="ffffff"><%=StringUtil.chanageNull(gb.
            getEmail(), "没填")%></td>
        </tr><tr>
            <td bordercolor="ffffff">
                主题:</td>
            <td bordercolor="ffffff"><%=gb.getTitle()%></td>
        </tr><tr>
            <td valign="top" bordercolor="ffffff">
                内容:</td>
            <td  valign="top"  bordercolor="ffffff"><%=StringUtil.chanage
            Null(gb.getContent(), "没填")%></td>
        </tr><tr>
            <td bordercolor="ffffff">
```

```
                时间:</td>
            <td bordercolor="ffffff"><%=gb.getTime()%></td>
        </tr>
    </table>
    <br>
    <%}}%>
</center>
...
```

12.3.3　多学两招——转向的两种方法

在 JSP 语法中可以通过 forward()或 sendRedirect()方法来实现转向功能，即从 Servlet 程序转向 JSP 页面或从 JSP 页面转向 Servlet 程序。那么两个方法是否完全相同可以实现互换呢？

下面通过底层的实现机制，来讲解一下两个方法如何实现由 Servlet 程序转向 JSP 页面。

forward()：首先浏览器调用 Servlet，Servlet 会在服务器端直接转向 JSP，然后 JSP 回应给浏览器，整个过程经历了 3 个步骤，如图 12.15 所示。

sendRedirect()：首先浏览器调用 Servle，Servlet 直接回应给浏览器，浏览器接着再次调用 JSP，最后 JSP 回应给浏览器。整个过程经历了 4 个步骤，如图 12.16 所示。

最后，通过表（如表 12.1 所示）来介绍两个方法的其他区别。

表 12.1　区别

区　　别	forward()	sendRedirect()
是否能读取转向前 request 设定的属性值	是	否
转向后地址栏显示的地址	原 Servlet 的地址	所要转到地址
是否可以转向到本地 Web 应用之外的页面或网站	否	是
转向的速度	快	慢

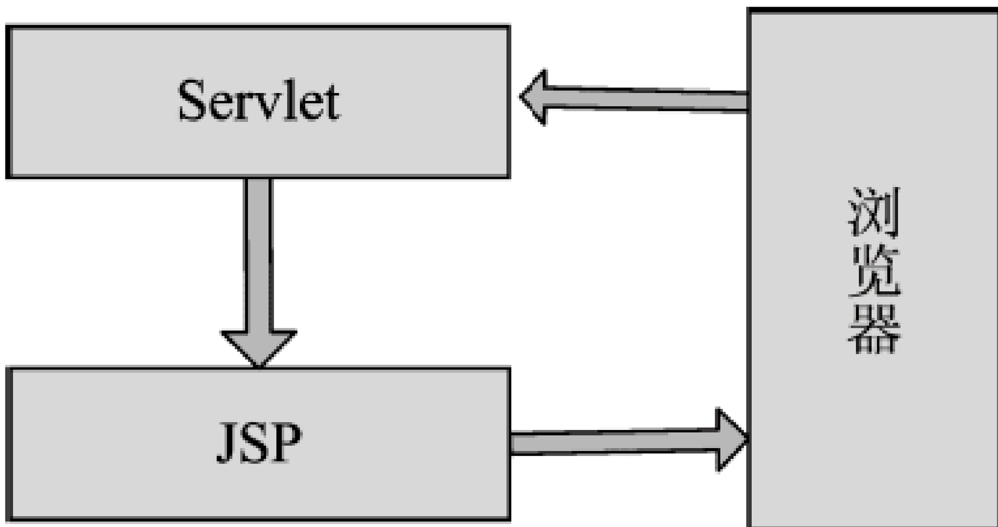


图 12.15　forward 转向机制

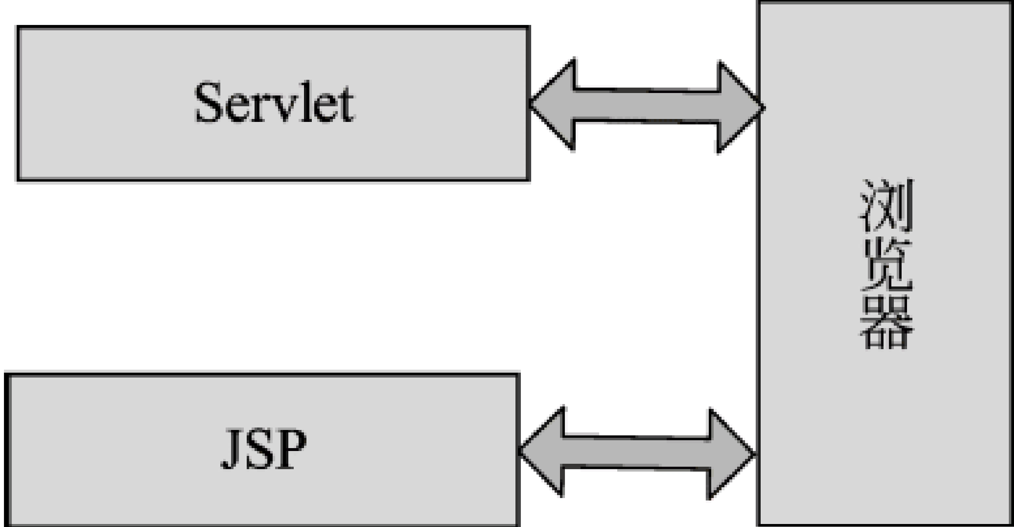


图 12.16　sendRedirect 转向机制

12.4　管 理 留 言

本节通过 JSP+Servlet+JavaBean 框架技术来实现留言管理功能，只有具有一定权限的用户才能对留言进行管理，管理留言程序流程如图 12.17 所示。

（1）浏览者通过登录页面来实现登录后，登录信息会被服务器端名叫 AdminUser-Servlet.java 的 Servlet 程序，来检验该浏览者是否具有管理权限。

（2）如果想管理留言内容，浏览器发送的请求资源是 list.jsp。该请求会经过名为

AuthenticationFilter.java 的过滤器，实现只有具有管理权限的浏览者才能访问 list.jsp 资源。

（3）当显示出 list.jsp 页面后，如果想实现“编辑”和“删除”操作时，会由服务器端名为 ManageServlet.java 的 Servlet 程序来实现相应的操作。

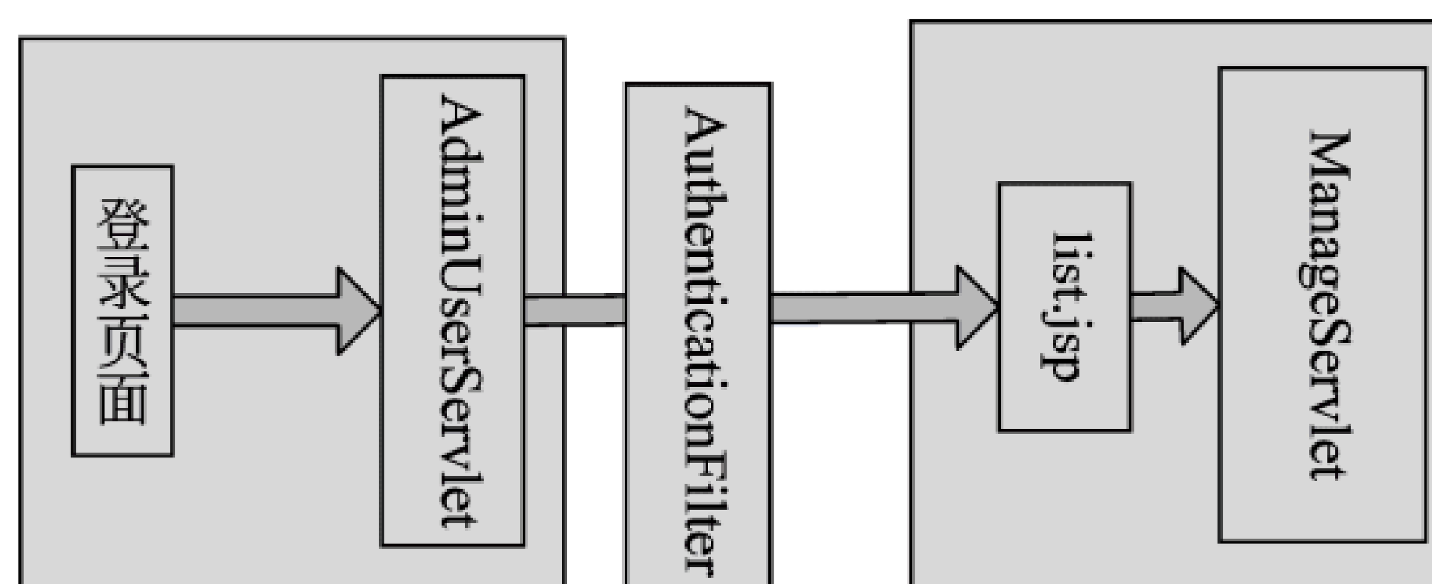


图 12.17　程序关系图

（4）为了防止出现乱码问题，所有的请求都需要经过过滤器 CharacterEncodingFilter.java 来处理字符编码问题。

12.4.1　用户登录页面和登录失败页面

login.jsp 和 loginFail.jsp 这两个页面分别是用户登录页面和登录失败后的页面。只有登录成功的用户才能实现对留言内容的管理。代码 12.7 用来设计登录界面。

代码 12.7　登录界面：login.jsp

```
...
<style>
* { font-family: "宋体"; font-size: 14px }
</style>
<script                                              type="text/javascript"
src="${ctx}/js/validation-framework.js"></script>
<p align="center">
    请您输入用户名与密码
</p>
<form  id="form2"  name="form2"  method="post"action="${ctx}/admin/user"
onsubmit="return doValidate(this)">
    <input type="hidden" name="q" value="login">
    <table width="500"  border="0" align="center">
        <tr><td width="100">用户名：</td>
        <td width="400">
        <input name="username" type="text" id="username" size="20" ></td>
        </tr>
        <tr><td>密码：</td> <td>
        <input name="password" type="password" id="password" size=
        "20"></td>
        </tr><tr>
        <td></td><td>
                <input type="submit" name="Submit" value="提交" >
                <input type="reset" name="Reset" value="重置" >
        </td></tr>
    </table>
</form>
...
```

当用户登录失败后，就会转向登录失败页面。代码 12.8 用来设计失败页面。

代码 12.8　登录失败界面：loginFail.jsp

```
...
<body>
```

```
对不起，输入的用户名与密码不正确。<br>
<a href="${ctx}/admin/login.jsp">重新登录</a>
</body>
...
```

12.4.2　对管理员的登录处理

只有拥有特定权限用户才能称为管理员，对于该种用户只有登录成功后，才能对留言内容进行管理。AdminUserServlet 是服务器端的 Servlet 程序，用来实现对管理员进行登录处理，具体内容如代码 12.9 所示。

代码 12.9　实现登录：AdminUserServlet.java

```
...
public class AdminUserServlet extends HttpServlet {
    private static final long serialVersionUID = 5801558969966197290L;
    //登录管理方法
    public  void  login(HttpServletRequest  request,  HttpServletResponse
    response) throws ServletException, IOException {
        String message = "";                          //定义了结果变量
        String username = request.getParameter("username");
                                                      //获取参数 username 的值
        String password = request.getParameter("password");
                                                      //获取参数 password 的值
        //判断用户与密码是否为空
        if (StringUtil.validateNull(username)) {
            message = "对不起，姓名不能为空，请您重新输入！<br>";
        } else if (StringUtil.validateNull(password)) {
            message = "对不起，密码不能为空，请您重新输入！<br>";
        } else {
            String param[] = { username, password };   //组成参数数组
            //创建数据库工具类
            OracleTool db = new OracleTool("java:/comp/env/jdbc/
            oracleds");
            db.init();                                //初始化工具类
            //从数据库中查找符合特定的记录，并返回一条记录转换成 MapLIst 类型的对象
            List result = (List) db.query("select id from admin where
            username=? and password=?", param,new MapListHandler());
            if (result.size() == 0) {                 //当变量 result 的大小为 0
                message = "对不起，用户名或者密码错误";
                request.setAttribute("guesbook.admin.login.message",
                message);
                request.getRequestDispatcher("/admin/loginFail.jsp").
                forward(request,response);
            } else {                              //当变量 result 的变量不为 0
                HttpSession session = request.getSession();
                                                  //创建一个 Session 对象
                //设置 Session 的属性
                session.setAttribute("guesbook.admin.username",
                username);
    //转向映射路径为/admin/secure/manage 的程序，并传递变量的值给该 Servlet 程序
    response.sendRedirect(request.getContextPath()+"/admin/secure/manage
    ?q=list");
            }
```

```
            }
        }
        //退出方法
        public void logout(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
            HttpSession session = request.getSession();    //获取 Session 对象
            //移除名为特定值的 Session 对象
            session.removeAttribute("guesbook.admin.username");
            //转向登录页面

        response.sendRedirect(request.getContextPath()+"/admin/login.jsp");
        }
        public void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
            String method = request.getParameter("q"); //接受参数 q 的值
            if (method != null && method.equals("login")) {
                                                        //判断后，调用登录方法
                login(request, response);
            } else {
                logout(request, response);              //调用退出方法
            }
        }
        public void doPost(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
            doGet(request, response);
        }
}
```

【代码解析】

- 在登录 login()方法中，首先通过数据库工具类获取数据库中符合要求的用户和密码记录。在使用工具类的 query()方法时，第 1 个参数为查询数据的 SQL 语句变量。第 2 个参数为参数数组。第 3 个参数为实现一条记录是一个 MapList 对象的效果，用 BeanListHandler 对象来代替。
- 在登录 login()方法中获取查询数据库的结果 result 后，就会判断其的大小。当其为空时，创建一个值为 messsage 的变量，然后转向到登录失败页面并显示 message 的值。否则创建一个属性值为 username 的 Session 对象，然后转向留言管理页面并传递一个参数 q。
- 在退出 logout()方法中，首先获取当前 Session 对象，然后删除属性值为特定值的对象。最后，转向登录界面。
- 修改 doGet()方法，根据参数 q 的值来判断调用的是登录方法还是退出方法。

12.4.3　利用过滤器实现资源管理

当只有拥有特定权限的用户才能浏览特定的目录，即利用过滤器 AuthenticationFilter 来实现对特定目录的处理。AdminUserServlet 是服务器端的 Servlet 程序，用来实现对资源管理，具体内容如代码 12.10 所示。

代码 12.10　资源管理：AuthenticationFilter.java

```
...
public class AuthenticationFilter implements Filter {
    String url = "/";           //定义一个字符串变量，表示验证失败后转向的路径
```

```
    public void destroy() {
    }
    //编写 doFilter()方法
    public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException,
            ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
                                                //类型转换
        HttpServletResponse res = (HttpServletResponse) response;
                                                //类型转换
        HttpSession session = req.getSession();         //获取 Session 对象
        if (session.getAttribute("guesbook.admin.username") == null) {
                                        //判断是否有特定的 Session
            res.sendRedirect(req.getContextPath() + url);
                                        //转向失败后的路径
        } else {
            chain.doFilter(request, response);          //访问要求的路径
        }
    }
    //编写 init()方法
    public void init(FilterConfig config) throws ServletException {
        url = config.getInitParameter("url");
                                    //没有管理权限，访问受限制资源转向的路径
    }
}
```

接着在 web.xml 文件中实现对该过滤器类进行配置。

```
    <!--配置过滤器 AuthenticationFilter -->
    <filter>
        <filter-name>AuthenticationFilter</filter-name>
        <filter-class>webbook.MessageBook.AuthenticationFilter</filter-
        class>
        <init-param>
            <param-name>url</param-name>
            <param-value>/admin/login.jsp</param-value>
        </init-param>
    </filter>
    <!--配置过滤器 AuthenticationFilter 映射路径-->
    <filter-mapping>
        <filter-name>AuthenticationFilter</filter-name>
        <url-pattern>/admin/secure/*</url-pattern>
    </filter-mapping>
```

【代码解析】

- 首先修改 init()方法，在该方法中定义了当没有管理权限的浏览者访问不该访问的路径后所要转向的路径。
- 接着修改 doFilter()方法，在该方法中首先对 request 和 response 对象进行强制类型转换，接着再判断是否有属性为 guesbook.admin.username 的 Session 对象。如果有则说明该对象具有管理权限，则转向所要求路径；否则转向验证失败后的路径。

同时为了在该项目中解决乱码问题，利用一个名为 CharacterEncodingFilter 的过滤器来管理传送的字符编码，代码 12.11 用来实现字符编码。

代码 12.11　实现字符编码：CharacterEncodingFilter.java

```
...
public class CharacterEncodingFilter implements Filter {
    private FilterConfig config;            //定义一个 FilterConfig 类型变量
    private String encoding = "ISO8859_1"; //定义默认的编码格式
    public void destroy() {                 //编写 destroy()方法
        config = null;
    }
    public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain)
            throws IOException, ServletException {
        request.setCharacterEncoding(encoding);    //设置所有的请求编码
        chain.doFilter(request, response);         //其他过滤器的调用方法
    }
    public void init(FilterConfig config) throws ServletException {
        this.config = config;                      //获取配置信息
        encoding = config.getInitParameter("encoding");
                                            //根据配置信息给变量 encoding 赋值
    }
}
```

接着在 web.xml 文件中实现对该过滤器类进行配置。

```
    <!--配置 EncodingFilter 过滤器-->
    <filter>
        <filter-name>EncodingFilter</filter-name>
        <filter-class>webbook.chapter15.CharacterEncodingFilter</filter-
        class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
    <!--配置 EncodingFilter 过滤器映射-->
    <filter-mapping>
        <filter-name>EncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

【代码解析】

- ❑ 首先定义两个变量，FilterConfig 类型的变量 config 用来获取配置信息，String 类型的变量 encoding 用来表示编码格式。
- ❑ 接着修改 init()方法，在该方法中首先读取配置文件，然后通过 getInitParameter ("encoding")方法把配置好的编码方式赋值给变量 encoding。
- ❑ 然后修改 doFilter()方法，在该方法中首先通过 request 对象的 setCharacterEncoding()方法设置所有请求的编码方式，然后再让其他过滤器调用该方法。
- ❑ 最后修改 destroy()方法，在该方法中对 config 对象赋为空。

12.4.4　管理留言内容

管理留言内容，即编辑留言内容、删除留言内容，这些操作主要通过 ManageServlet.java 文件来实现。ManageServlet 类是服务器端的 Servlet 程序，用来实现对留言内容进行特定

的操作，具体内容如代码 12.12 所示。

代码 12.12　管理留言内容：ManageServlet.java

```
...
public class ManageServlet extends HttpServlet {
    OracleTool db = null;
    //编辑方法
    public  void  edit(HttpServletRequest  request,  HttpServletResponse
    response) throws ServletException, IOException {
        String params[] = { request.getParameter("id") };
                                                    //获取参数 ID 的值
        String sql = "select * from MessageBook where id=?";
                                                    //设置一个 SQL 字符串
        //从数据库中查找符合特定的记录，并返回一条记录转换成 LIst 类型的对象
        List list = (List) db.query(sql, params, new BeanListHandler(Message
        Book.class));
        //把 list 的值存储起来
        request.setAttribute("MessageBook.admin.edit", list.get(0));
        //转向到编辑页面
        request.getRequestDispatcher("/admin/secure/edit.jsp").forward
        (request, response);
    }
    //更新方法
    public  void  update(HttpServletRequest  request,  HttpServletResponse
    response) throws ServletException, IOException {
        //获取专递过来的参数
        String params[] = { request.getParameter("name"), request.get
        Parameter("email"), request.getParameter("phone"),
                request.getParameter("title"), request.getParameter
                ("content"), request.getParameter("id") };
        //通过工具类的 update()方法实现更新
        int i = db.update("update MessageBook set name=?,email=?,phone=?,
        title=? ,content=? where id=?", params);
        //根据更新结果判断
        if (i == 1) {
            //设置属性
            request.setAttribute("MessageBook.admin.update.message",  "更
            新成功！");
            MessageBook gb = new MessageBook();//创建 MessageBook 对象
            //设置 gb 的值
            gb.setId(Integer.parseInt(request.getParameter("id")));
            gb.setName(request.getParameter("name"));
            gb.setEmail(request.getParameter("email"));
            gb.setPhone(request.getParameter("phone"));
            gb.setTitle(request.getParameter("title"));
            gb.setContent(request.getParameter("content"));
            //设置属性
            request.setAttribute("MessageBook.admin.edit", gb);
        } else {
            //设置属性
            request.setAttribute("MessageBook.admin.update.message",  "更
            新失败！");
        }
    //转向更新结果页面
    request.getRequestDispatcher("/admin/secure/updateResult.jsp").forwa
    rd(request, response);
```

```
    }
    //删除方法
    public void delete(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        //获取传递过来的参数
        String params[] = { request.getParameter("id") };
        //通过工具类的 update()方法实现更新
        int i = db.update("delete from MessageBook where id=?", params);
        if (i == 1) {
            request.setAttribute("MessageBook.admin.delete.message", "删
            除成功");
        } else {
            request.setAttribute("MessageBook.admin.delete.message", "删
            除失败");
        }
    request.getRequestDispatcher("/admin/secure/deleteResult.jsp").forwa
    rd(request, response);
    }
    //显示所有留言内容
    public void list(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        //从数据库中查找符合特定的记录，并返回一条记录转换成 LIst 类型的对象
        List list = (List) db.query("select id,name,title from MessageBook
        order by id desc", null, new BeanListHandler(
                MessageBook.class));
        request.setAttribute("MessageBook.admin.list", list);
        request.getRequestDispatcher("/admin/secure/list.jsp").forward
        (request, response);
    }
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        db = new OracleTool("java:/comp/env/jdbc/oracleds");
                                              //创建 Oracle 数据库工具类
        db.init();
        String method = request.getParameter("q"); //获取参数 q 的值
        if(method==null){                          //当 q 为空时
            method="list";                         //设置 method 的值为 list
        }
        if (method.equals("edit")) {               //当 method 为 edit
            edit(request, response);
        } else if (method.equals("delete")) {      //当 method 为 delete
            delete(request, response);
        } else if (method.equals("update")) {      //当 method 为 update
            update(request, response);
        } else {
            list(request, response);
        }
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        doGet(request, response);
    }
}
```

接着在 web.xml 文件中实现对该类进行配置。

```
<!--配置 ManageServlet 类路径-->
<servlet>
```

```
    <servlet-name>ManageServlet</servlet-name>
    <servlet-class>webbook.MessageBook.ManageServlet</servlet-class>
  </servlet>
  <!--配置 ManageServlet 映射路径 -->
  <servlet-mapping>
    <servlet-name>ManageServlet</servlet-name>
    <url-pattern>/admin/secure/manage</url-pattern>
  </servlet-mapping>
```

【代码解析】

- 在 doGet()方法中首先创建了数据库工具类 db，然后定义 method 变量来获取传递过来 q 参数的值。最后判断变量 method 的值，在具体判断之前先设置该变量的值默认为 list。当变量 method 值为 edit 时，调用 edit()方法；当变量 method 值为 delete 时，调用 delete()方法；当变量 method 值为 update 时，调用 update ()方法；其他值一律调用 list()方法。
- 在 update()方法中首先定义一个 params 变量，用来存储传递过来的参数值，接着利用数据库工具类的 update()方法更新数据库，该方法会返回一个结果。利用返回的结果判断更新是否成功，当更新成功后，就利用传递过来的参数创建一个 MessageBook 对象，然后利用 request 对象的方法把该记录和“更新成功”传递到 updateResult.jsp 页面显示；当更新失败后，会利用 request 对象的方法把“更新失败”传递到 updateResult.jsp 页面显示。
- 在 delete()方法中首先定义一个 params 变量，用来存储传递过来的 ID 的值，接着利用数据库工具类的 update()方法更新数据库，该方法会返回一个结果。利用返回的结果判断更新是否成功，当更新成功后，就会利用 request 对象的方法把该“删除成功”字符串传递到 deleteResult.jsp 页面显示；当删除失败后，会利用 request 对象的方法把“删除失败”字符串传递到 deleteResult.jsp 页面显示。
- 在 list()方法中首先会利用数据库工具类的 query()方法把数据库所有记录转变成 MessageBook 对象，然后存储到数组 params 中，最后利用 request 对象的方法把所有记录传递到 list.jsp 页面显示。

12.4.5　管理留言所需要的页面

在进行留言管理时，会经历这样的一个过程（如图 12.18 所示）：当浏览者成功登录后，会显示 list.jsp 页面。如果想进行编辑操作，单击“编辑”链接就会进入 edit.jsp 页面，在该页面进行相应的修改后，就会进入 updateResult.jsp 页面来显示编辑的结果。如果想进行删除操作，单击“删除”链接就会进入 deltteResult.jsp 页面来显示删除结果。

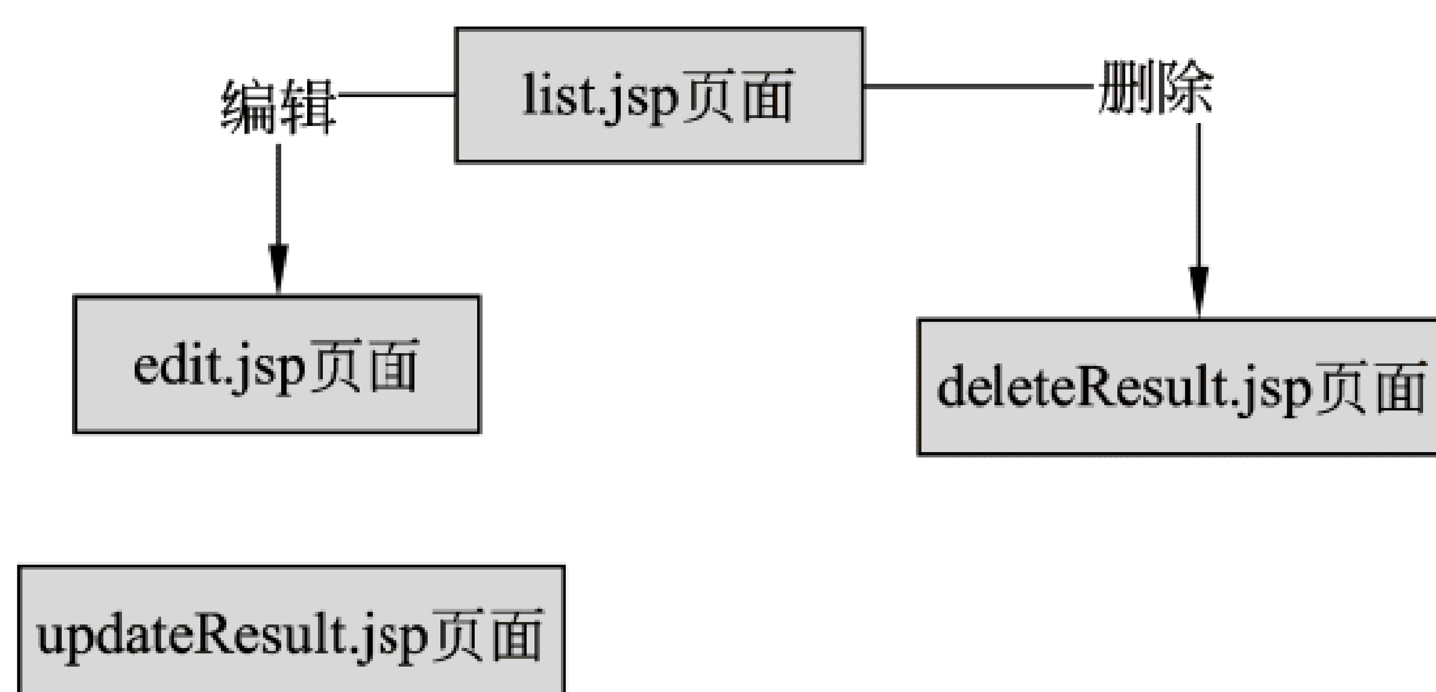


图 12.18　程序关系

代码 12.13 用来设计显示管理留言的页面，在该页面中可以对留言进行编辑和删除操作。代码 12.14 用来设计编辑留言的页面，在该页面可以对相应的留言进行修改，修改完

成后就会直接转入代码 12.15 设计的显示修改结果的页面。代码 12.16 用来设计显示删除结果的页面。

代码 12.13　管理留言页面：list.jsp

```
...
<center>
    <a href="${ctx}/admin/user?q=logout">退出登录</a><br><br>
    所有留言内容<br><br>
    <!--遍历传递过来的 MessageBook-->
    <c:forEach items="${requestScope['MessageBook.admin.list']}"  var=
    "gb">
    <table>
...
        <td>留言编号: </td>
        <td >${gb.id}</td>
        <td>作者:</td>
        <td><c:out value="${gb.name}" default="没填" /></td>
        <td>主题: </td>
        <td><c:out value="${gb.title}" default="没填" /></td>
        <td>操作选项: </td>
            <a href="${ctx}/admin/secure/manage?q=edit&id=${gb.id}">编辑
            </a>
            <a href="${ctx}/admin/secure/manage?q=delete&id=${gb.id}">删
            除</a></td>
        </tr>
...
    </table>
    <br>
    </c:forEach>
</center>
...
```

代码 12.14　编辑留言页面：edit.jsp

```
...
<p align="center">修改留言</p>
    <a href="${ctx}/admin/secure/manage?q=list">返回管理首页</a>|<a href=
    "${ctx}/admin/user?q=logout">退出登录</a>
    <!--创建一个变量，其值为传递过来的 MessageBook-->
    <c:set var="gb" value="${requestScope['MessageBook.admin.edit']}" />
    <!--表单-->
    <form  id="form1"  name="form1"  method="post"action="${ctx}/admin/
    secure/manage">
    <!--传递参数 ID 与 q 的值-->
    <input type="hidden" name="id" value="${gb.id}">
    <input type="hidden" name="q" value="update">
    <table>
...
        <td>姓名: </td>
        <input name="name" type="text" id="name" size="40"  value="${gb.
        name}" >
        <td>E-Mail: </td>
        <input name="email" type="text" id="email" size="40" value="${gb.
        email}" >
        <td>电话: </td>
```

```
        <input name="phone" type="text" id="phone" size="40"  value="${gb.
        phone}" >
        <td>主题：</td>
        <input name="title" type="text" id="title" size="80"  value="${gb.
        title}" >
        <!--FCKeditor 编辑器-->
        <script type="text/javascript">
            window.onload = function(){
                var oFCKeditor = new FCKeditor( 'content' ) ;
                oFCKeditor.BasePath = '${ctx}/fckeditor/' ;
                oFCKeditor.ToolbarSet = 'Basic';
            oFCKeditor.Height = 300 ;
                oFCKeditor.ReplaceTextarea() ;
            }
        </script>
        <td valign="top">内容：</td>
        <textarea name="content" rows="10" cols="80">${gb.content}
        </textarea>
        <!--提交按钮-->
        <td>
            <input type="submit" name="Submit" value="提交" />
            <input type="reset" name="Reset" value="重置" />
        </td>
...
    </table>
</form>
...
```

代码 12.15　显示编辑结果页面：updateResult.jsp

```
...
<center>
    <!--输出更新成功与否的信息-->
    <c:out
value="${requestScope['MessageBook.admin.update.message']}"/><br>
    <!--创建一个变量，其值为传递过来的 MessageBook-->
    <c:set var="gb" value="${requestScope['MessageBook.admin.edit']}" />

    <!--利用表格输出 gb 对象属性的值-->
    <table>
...
        <td>编号:</td>
        <td><c:out value="${gb.id}"/></td>
        <td>姓名:</td>
        <td><c:out value="${gb.name}"/></td>
        <td>电话:</td>
        <td><c:out value="${gb.phone}" default="没填" /></td>
        <td>email: </td>
        <td><c:out value="${gb.email}" default="没填" /></td>
        <td>主题:
        <td><c:out value="${gb.title}" default="没填" /></td>
        <td>内容:   </td>
        <td><c:out value="${gb.content}" default="没填" escapeXml=
        "false"/></td>
        <td>时间:</td>
        <td ><c:out value="${gb.time}"/></td>
...
```

```
    </table>
    </center>
    <br><a href="${ctx}/admin/secure/manage?q=list">返回管理首页</a><br>
...
```

代码 12.16　显示删除结果页面：deleteResult.jsp

```
...
<!--输出删除成功与否的信息-->
<center>
    <c:out
value="${requestScope['MessageBook.admin.delete.message']}"/><br>
    <a href="${ctx}/admin/secure/manage?q=list">返回管理首页</a><br>
</center>
...
```

12.4.6　多学两招——过滤器介绍

过滤器跟现实中的污水净化设备一样，用来实现源数据和目的数据之间的过滤。对于 Java Web 应用程序来说，过滤器其实就是对客户端和资源之间的请求与响应信息进行过滤的服务器端 Web 组件。

当一个 Java Web 应用程序拥有一个过滤器时（如图 12.19 所示），将如何处理浏览器发出请求并发送给浏览器的响应信息？

当 Web 容器收到一个资源的请求时，首先会判断该资源是否与过滤器有关联。如果存在关联，该 Web 容器就会把请求交给过滤器处理。经过过滤器的处理后，最后请求才会被发送给目标资源。同理，当目标资源对请求作出响应后，该响应同样会先被过滤器处理，然后才会被发送给浏览器。

注意： 在具体开发时，一个 Java Web 应用程序可能具有不止一个过滤器。当具有多个过滤器时就会被称为过滤器链（如图 12.20 所示）。

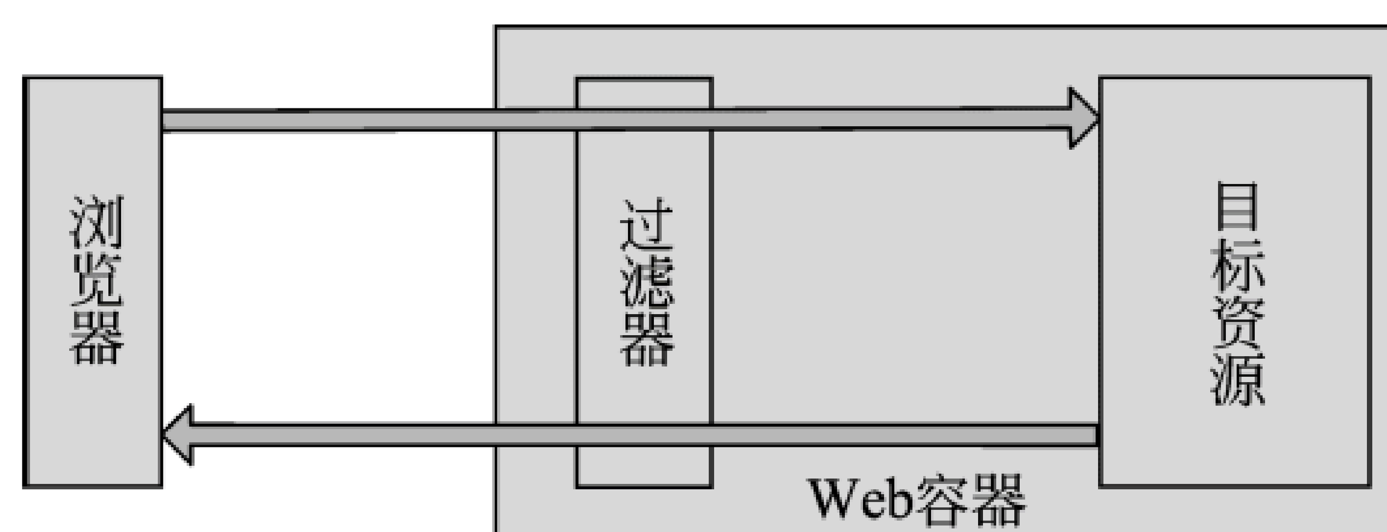


图 12.19　单个过滤器

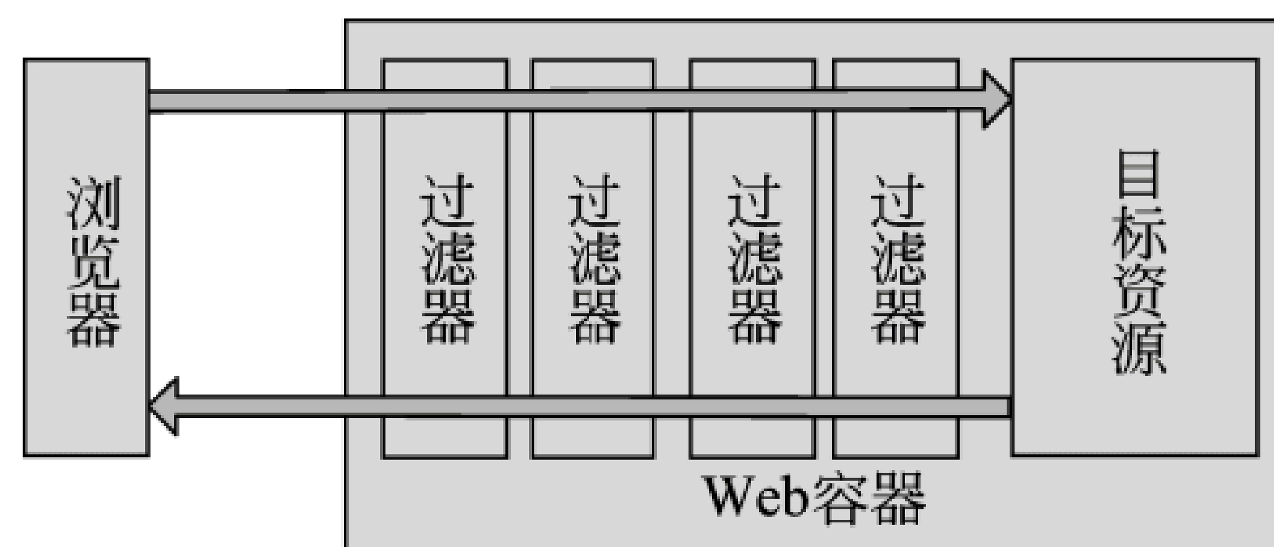


图 12.20　过滤器链

经过过滤器处理后的请求下一步骤可能是：发送给另一个过滤器；发送给所请求的目标资源；发送给另一个目标资源；发送给浏览器。

在 Java Web 开发中，过滤器主要用来实现如下应用。

- 用户认证与授权管理。
- 统计 Java Web 应用的访问量和访问的命中率，形成访问报告。
- 实现 Web 应用的日志处理功能。
- 实现图像格式转换。
- 实现数据压缩功能。

- 对传输的数据进行加密。
- 触发资源访问事件。
- 实现 XML 文件的 XSLT 转换。

12.4.7　多学两招——过滤器开发和部署

在 Servlet 中要开发过滤器，必须了解 3 个接口：javax.servlet.Filter 接口、javax.servlet.FilterConfig 接口和 javax.servlet.FilterChain 接口。创建一个过滤器，需要两个步骤：创建一个继承 Filter 接口的类和配置该过滤器。

在 MyEclipse 开发环境中创建一个继承 Filter 的接口类 test，具体内容如代码 12.17 所示。

代码 12.17　继承 Filter 接口：test.java

```
//导入包
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class test implements Filter {                   //编写 test 类
    public void destroy() {                             //编写 destroy()方法
    }
    //编写 doFilter()方法
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
    }
    public void init(FilterConfig filterConfig) throws ServletException
    {                                                   //编写 init()方法
    }
}
```

【代码解析】

- 在该 java 文件引入的包中有 3 个非常重要的包：import javax.servlet.Filter、import javax.servlet.FilterChain 和 import javax.servlet.FilterConfig。
- init()方法为实现 Filter 接口中的方法，用来实现初始化过滤器。Java Web 容器在调用该方法时会传递进来 FilterConfig 类型对象。利用该对象获取在部署描述符中配置的过滤器的初始化参数，就可以达到初始化过滤器的目的。最后 init()方法可以通过抛出 ServletException 异常，来通知 Java Web 容器过滤器不能正常工作。
- init()方法中拥有一个用来向过滤器初始化时传递信息的 FilterConfig 类型参数，该参数的类型全称为 javax.servlet.FilterConfig。FilterConfig 由 Java Web 容器实现，然后将其实例传入到过滤器对象的 init()方法中。最后，在 init()方法中通过调用 FilterConfig 实例的方法（getFilterName、getFilterInitParameter、getFilterInitParameterNames 和 getServletContext）来初始化过滤器。
- doFilter()方法为过滤器的核心方法，用来对请求和响应进行特定的处理，实现过滤器的核心逻辑。当浏览器请求目标资源时，Java Web 容器就会调用与该目标资源相关联的过滤器的 doFilter()方法，然后实现该方法中的特定操作。

- ❑ doFilter()方法拥有 3 个参数：ServletRequest、ServletResponse 和 FilterChain。前两种类型的参数对应发送的请求和返回的响应，同时要注意这些请求与响应是与具体协议无关。而最后一个 FilterChain 类型的参数主要是为了达到顺畅连续的调用过滤器目的，即执行完一个过滤器（如图 12.21 所示），再调用紧跟其后的另一个过滤器。FilterChain 接口全称 javax.servlet.FilterChain，而且只有一个 doFilter()方法。当过滤器对象调用该接口的 doFilter()方法时，就会使过滤器链中的下一个过滤器被调用。
- ❑ destroy()方法用来结束过滤器的生命周期，在该方法中可以编写销毁过滤器前的代码，即释放过滤器使用的资源。

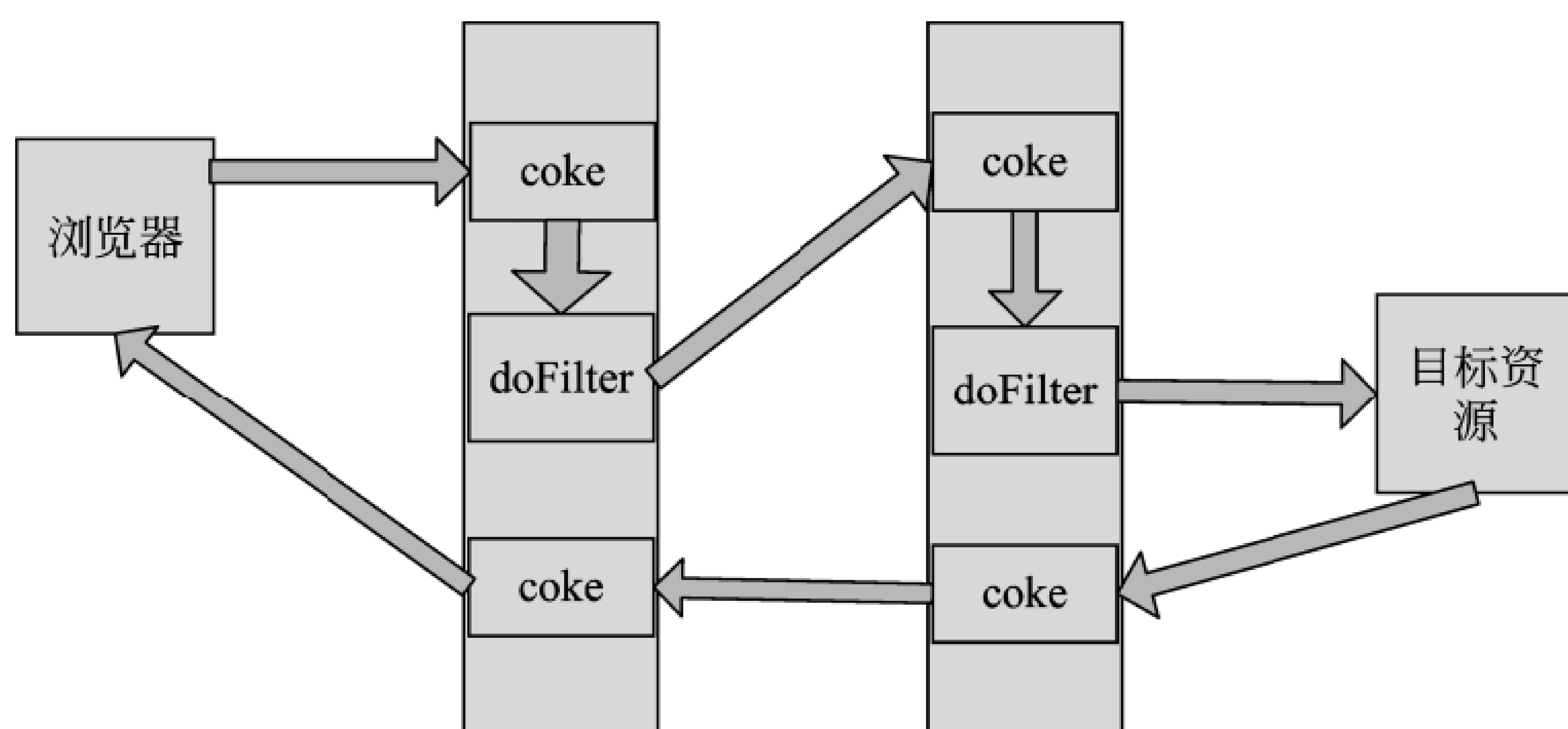


图 12.21　过滤器链的工作流程

注意：过滤器对象虽然是通过调用 chain.doFilter()方法将请求传给下一个过滤器，但是却通过调用 RequestDispatcher.forward()或 HttpServletResponse.sendRedirect()方法将请求转向其他资源。

编写完过滤器后，接着还需要通过在 web.xml 文件中对其进行配置。在具体配置时，需要用到两个元素：<filter>和<filter-name>。

<filter>元素用于在 Java Web 应用程序中声明一个过滤器，其结构如图 12.22 所示，每个元素的具体作用如表 12.2 所示。<filter-name>元素用于将声明的过滤器与 Servler 或 URL 模式相关联，其结构如图 12.23 所示，每个元素的具体作用如表 12.3 所示。

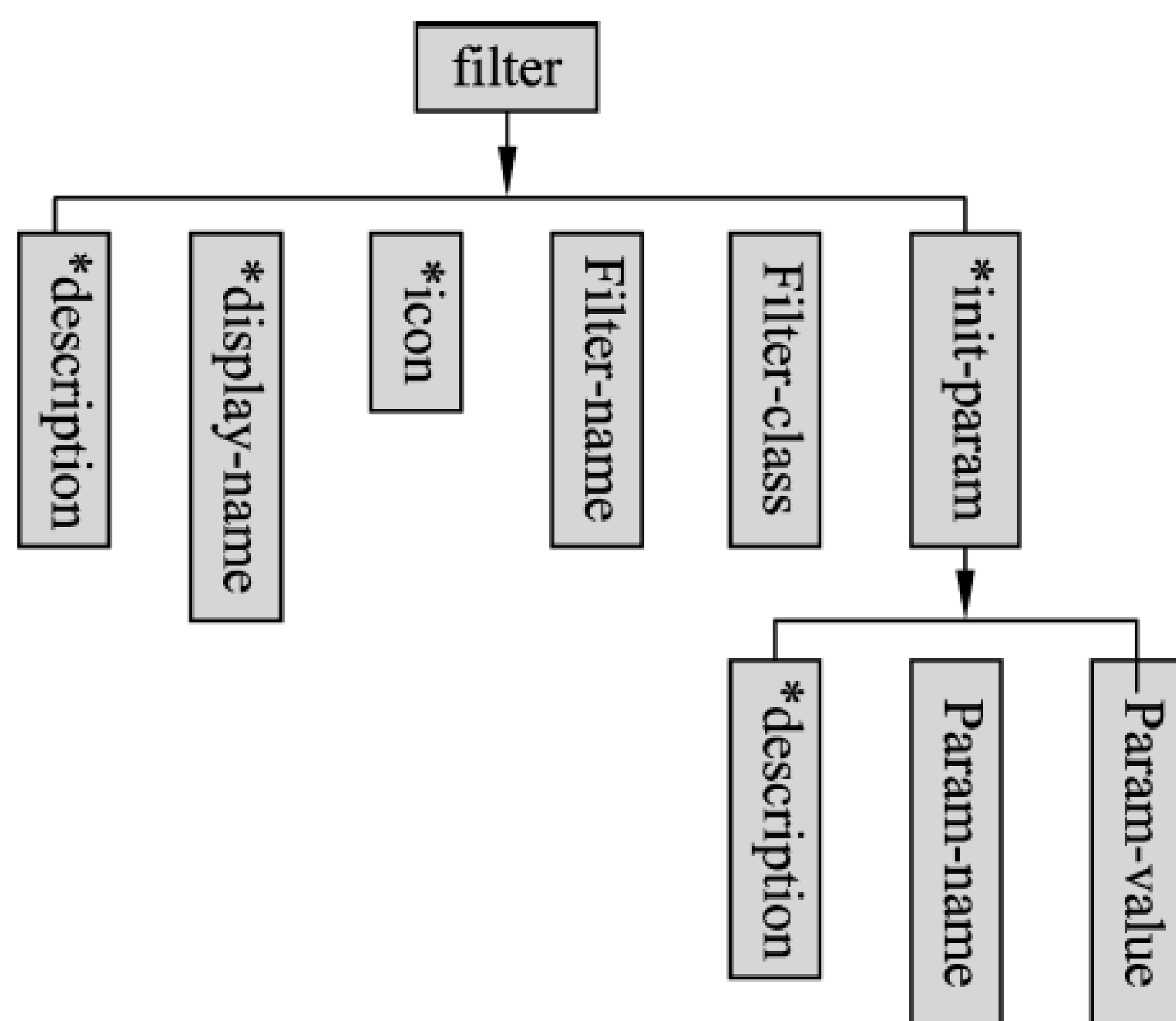


图 12.22　<filter>元素结构

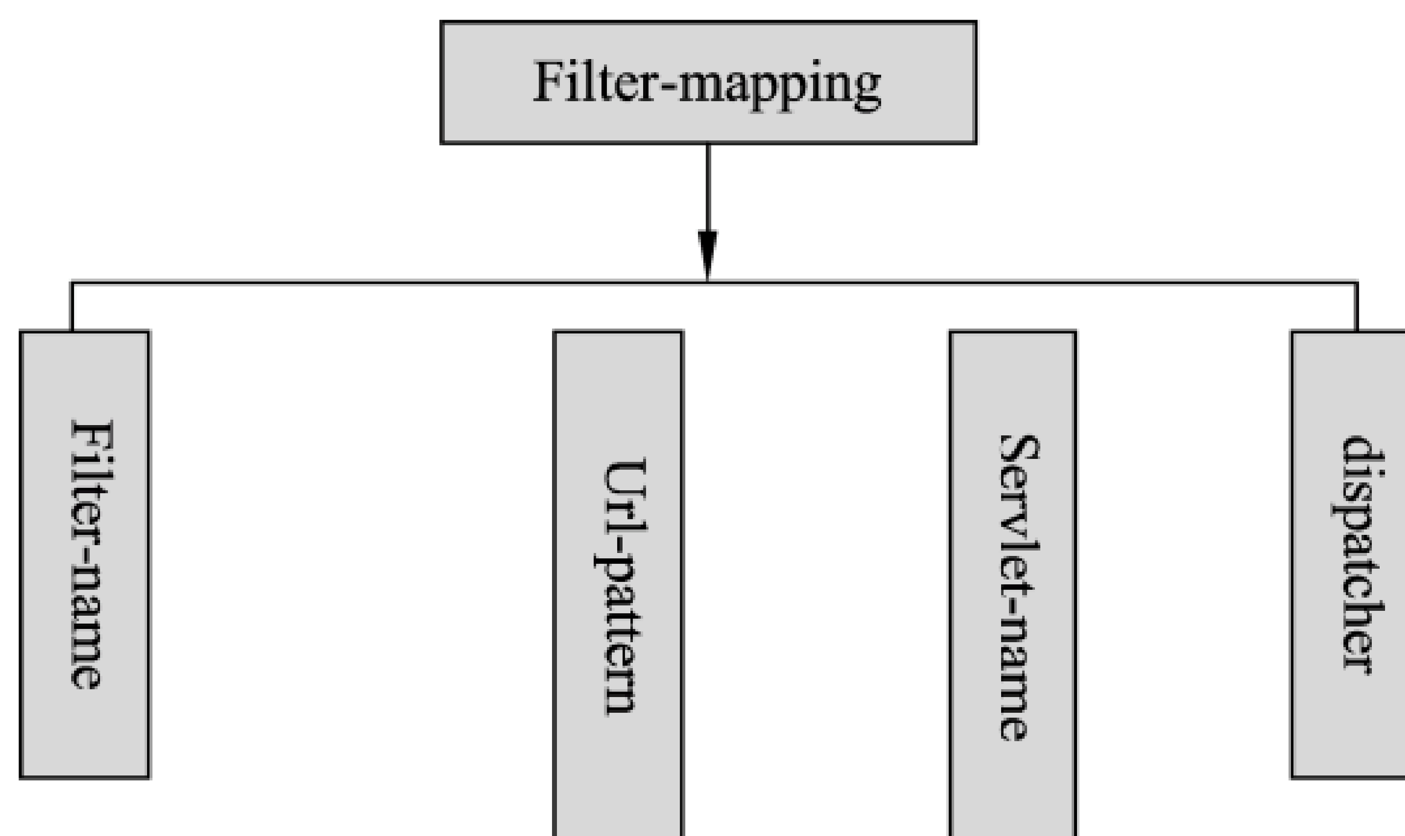


图 12.23　<filter-name>元素结构

表 12.2　<filter>元素

元　　素	作　　用
Description	指定过滤器的一个文本描述
Display-name	指定过滤器的一个简单名字
Icon	指定过滤器的一个图标
Filter-name	指定过滤器的一个名字
Filter-class	指定过滤器的完整路径
Init-param	指定过滤器初始化参数
Description	指定参数的一个文本描述
Param-name	指定初始化参数的名字
Param-value	指定初始化参数的值

表 12.3　<filter-name>元素

元　　素	作　　用
Filter-name	其值必须是在<filter>元素中声明过的过滤器名字
Filter-class	指定与过滤器相关联的 URL
servlet-param	指定与过滤器相对应的 Servlet
Dispatcher	指定过滤器的请求方式

12.5　使用 DAO 模式网络留言板

为了让项目的结构更清晰，本节将利用 DAO 模式来重新编写网络留言板。虽然使用 DAO 模式意味着比以前创建更多的类，但是却会使用项目更容易移植、阅读。

12.5.1　DAO 模式介绍

DAO（Data Access Object）全称数据访问接口，实际上是由 Data Accessor 模式和 Active Domain Object 模式这两个模式组成，其中 Data Accessor 模式实现了数据访问和业务逻辑的分离，而 Active Domain Object 模式实现了业务数据的对象化封装。

DAO 主要用来处理如下问题：由于持久性存储是各不相同的，所以访问不同持久性存储时的 API 也不同，因此当 J2EE 应用程序在不同的持久性存储间迁移时，访问持久存储层的代码就需要重写。为了解决这个问题，可以在 J2EE 应用程序中使用 DAO，将底层数据访问逻辑和业务逻辑分离开来，为不同数据源提供 CRUD 操作的 DAO 类，如图 12.24 所示。

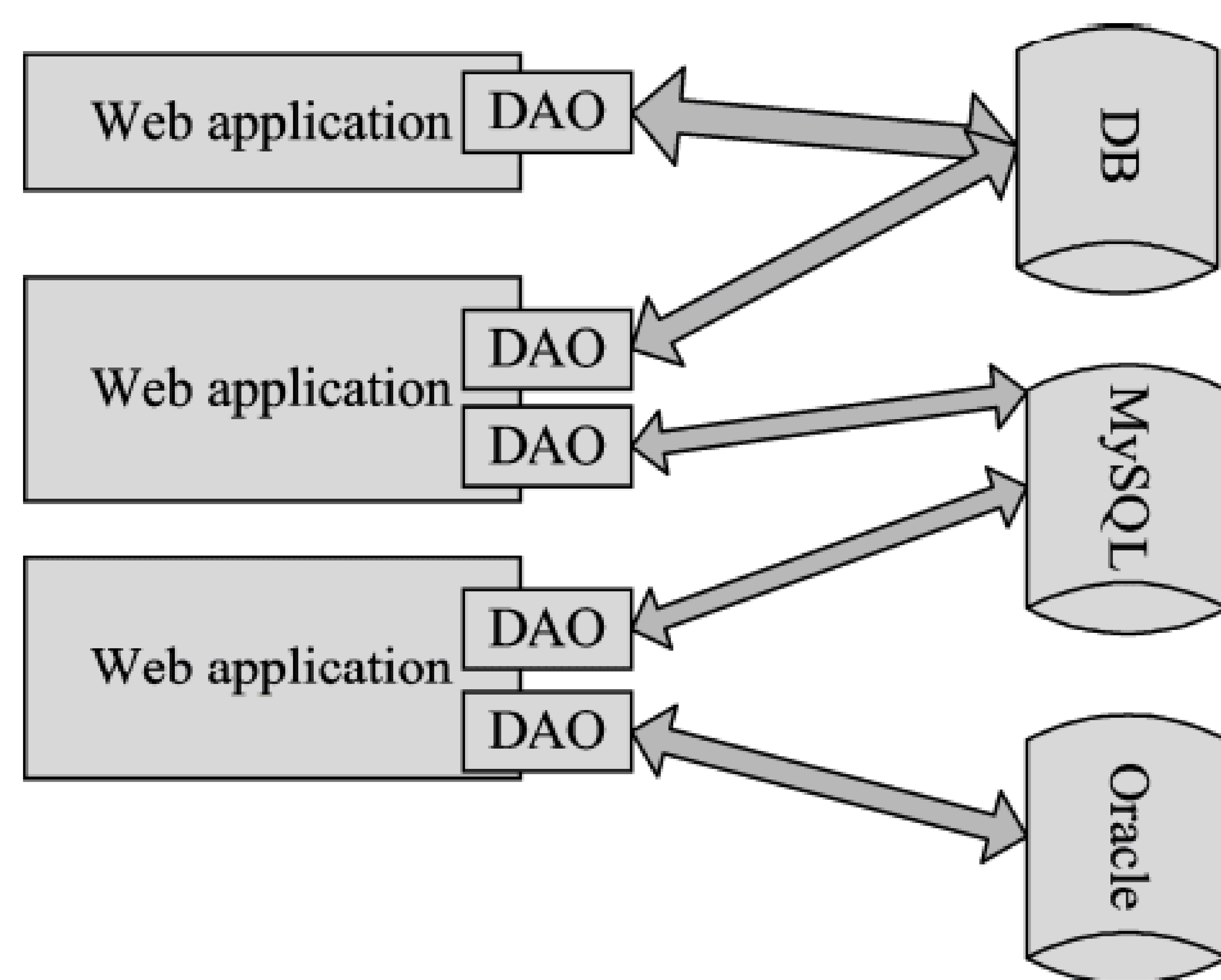


图 12.24　解决方案

注意：所谓的持久性存储就是存储持久性数据，例如数据库、文件等。

DAO 模式是如何实现呢？

为了便于讲解，下面将利用网络留言板中的留言内容对象来说明该问题。首先针对持久性数据（数据库中的表 MessageBook）定义一个接口（MessageBookDAO），在该接口中定义对该表进行的各种操作方法，一般为 CRUD 的 4 种操作。

其次接口是不能实例化的，所以必须针对各种持久层存储实现接口（MessageBook-DAOJdbc）。在具体实现该接口时，一般会用到底层的 POJO（MessageBook）。

经过上述的两个步骤后，应用程序（AddMessageServlet）不直接操作 POJO（Message-Book），而是通过接口（MessageBookDAO）和接口实现类（MessageBookDAOJdbc）来间接地操作 POJO（MessageBook）。

注意：经过上述处理后，当需要更换不同的持久性存储时，只要更换一行代码就可以。或者如果加入 Sping 技术后，根本不需要更改代码。

可以这样理解 DAO，其实就是先将数据库表和对象对应，然后提供一系列简单的对象，最后这些对象通过简单的方法实现数据持久化。以达到向上层隐藏实现细节，并使得项目与具体数据库系统无关。

通过使用 DAO 模式，可以实现以下目标。

- 数据存储逻辑的分离：通过把数据访问逻辑抽象成数据访问接口，使某些精通数据库操作技术的开发人员可以根据接口，提供数据库访问的最优化实现，而精通业务的开发人员则可以抛开数据繁琐细节，专注于业务逻辑编码。
- 数据访问底层实现的分离：为了分离数据使用和数据访问，可以通过 DAO 模式将数据访问分为抽象层和实现层。
- 资源管理和调度的分离：DAO 模式通过将数据访问逻辑从业务逻辑中脱离开来，使得在数据访问层实现统一的资源调度成为可能，以达到资源管理和调度的分离。
- 数据抽象：DAO 模式通过对底层数据的封装，来实现数据的抽象。

12.5.2　针对留言内容 MessageBook

在本节中，将针对 MessageBook 这个 POJO 类，利用 DAO 模式实现对自己的 CRUD 操作。底层类 MessageBook.java 跟原来的一样，不需要进行修改。只需要新建两个类：MessageBookDAO.java 和 MessageBookDAOJdbc.java，代码 12.18 是用来实现对 Message-Book 进行 CRUD 操作的接口类，代码 12.19 是实现前面接口的类。

代码 12.18　CRUD 的接口类：MessageBookDAO.java

```
public interface MessageBookDAO {
    public MessageBook getMessageBook(Integer id);
                                        //根据 ID 号获取 MessageBook 对象
    public List getMessageBooks();      //获取所有的 MessageBook 对象
    public boolean save(MessageBook gb);    //对 MessageBook 对象进行保存
    public boolean update(MessageBook gb); //对 MessageBook 对象进行更新
    public boolean remove(Integer id);      //根据 ID 号删除 MessageBook 对象
}
```

代码 12.19　CRUD 接口的实现类：MessageBookDAOJdbc.java

```
public class MessageBookDAOJdbc implements MessageBookDAO {
    //设置各种 SQL 语句
    private static final String SELECT_MESSAGEBOOKS_SQL = "select * from
    MessageBook order by id desc";              //查找所有记录的 SQL 语句
    private static final String SELECT_MESSAGEBOOK_SQL = "select * from
    MessageBook where id=?";                    //查找特定 ID 号的记录
    private static final String SAVE_MESSAGEBOOK_SQL = "insert into
    MessageBook (id,name,email,phone,title,content,time) values(gb_seq.
    nextval,?,?,?,?,?,?)";
                                                //插入某个特定的记录
    private static final String UPDATE_MESSAGEBOOK_SQL = "update MessageBook
    set name=?,email=?,phone=?,title=? ,content=? where id=?";
                                                //更新特定 ID 号的记录
    private static final String DELETE_MESSAGEBOOK_SQL = "delete from
    MessageBook where id=?";                    //删除特定 ID 号的记录
    OracleTool db = null;                       //创建一个数据库工具类 db
    //构造函数
    public MessageBookDAOJdbc() {
        db = new OracleTool("java:/comp/env/jdbc/oracleds");
                                                //为构造函数赋值
        db.init();                              //初始化数据库工具类
    }
    //有参获取留言对象方法
    public MessageBook getMessageBook(Integer id) {
        String params[] = { id.toString() };
                                            //获取参数 ID，存储到 params 数组
        List list = (List) db.query(SELECT_MESSAGEBOOK_SQL, params, new
    BeanListHandler(MessageBook.class)); //利用工具类的 query()方法操作数据库
        return (MessageBook) list.get(0);  //输出查找到的 MessageBook 对象
    }
    //无参获取留言对象方法
    public List getMessageBooks() {
        return (List) db.query(SELECT_MESSAGEBOOKS_SQL, null, new BeanList
        Handler(MessageBook.class));
    }
    //编写删除方法
    public boolean remove(Integer id) {
        String params[] = { id.toString() };
                                            //获取参数 ID，存储到 params 数组
        //利用工具类的 update()方法操作数据库
        int result = db.update(DELETE_MESSAGEBOOK_SQL, params);
        if (result == 0) {                  //判断结果
            return false;
        }
        return true;
    }
    //编写添加方法
    public boolean save(MessageBook gb) {
        String param[] = {gb.getName(),gb.getEmail(),gb.getPhone(),gb.
        getTitle(), gb.getContent(),gb.getTime() };
                                            //获取参数，存储到 params 数组
        //利用工具类的 update()方法操作数据库
        int result = db.update(SAVE_MESSAGEBOOK_SQL, param);
        if (result == 0) {                  //判断结果
            return false;
```

```
        }
        return true;
    }
    //编写更新方法
    public boolean update(MessageBook gb) {
        String params[] = { gb.getName(), gb.getEmail(),gb.getPhone(), gb.
        getTitle(), gb.getContent(),gb.getId().toString() };
                                        //获取参数，存储到 params 数组
        //利用工具类的 update()方法操作数据库
        int result = db.update(UPDATE_MESSAGEBOOK_SQL, params);
        if (result == 0) {                      //判断结果
            return false;
        }
        return true;
    }
}
```

最后，修改 AddMessageServlet.java 和 GetMessagesServlet.java 操作类，代码 12.20 调用前面开发的两个类来实现添加留言内容。代码 12.21 调用前面开发的两个类实现获取留言的内容。

代码 12.20　添加留言：AddMessageServlet.java

```
public class AddMessageServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        //定义各种变量
        boolean result = false;
        String message = "";
        //获取各种参数
        String name = request.getParameter("name");
        String title = request.getParameter("title");
        if (StringUtil.validateNull(name)) {            //判断变量 name
            message = "对不起，姓名不能为空，请您重新输入！<br>";
        } else if (StringUtil.validateNull(title)) {    //判断变量 title
            message = "对不起，主题不能为空，请您重新输入！<br>";
        } else {
            MessageBook gb = new MessageBook();     //创建 MessageBook 对象
            //创建 MessageBookDAOJdbc 对象
            MessageBookDAO dao = new MessageBookDAOJdbc();
            gb.setName(StringUtil.filterHtml(name));
            gb.setTitle(StringUtil.filterHtml(title));
            gb.setPhone(StringUtil.filterHtml(request.getParameter
            ("phone")));
            gb.setEmail(StringUtil.filterHtml(request.getParameter
            ("email")));
            gb.setContent(request.getParameter("content"));
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:
            mm:ss");
            gb.setTime(sdf.format(new java.util.Date()));
            result = dao.save(gb);
            //判断变量 result
            if (result) {
                message = "祝贺您，成功添加留言。";
            } else {
                message = "对不起，添加留言不成功，请您重新输入！";
            }
```

```
        }
        request.setAttribute("message", message);              //设置属性
        //实现转向
        request.getRequestDispatcher("/addResult.jsp").forward(request,
            response);
    }
}
```

代码 12.21　获取留言：GetMessagesServlet.java

```
...
public class GetMessagesServlet extends HttpServlet {
    public  void  doGet(HttpServletRequest  request,  HttpServletResponse
    response) throws ServletException, IOException {
        //创建 MessageBookDAOJdbc 对象
        MessageBookDAO dao = new MessageBookDAOJdbc();
        List list=dao.getMessageBooks() ;              //获取所有留言内容
        request.setAttribute("results", list);
       request.getRequestDispatcher("/getMessages.jsp").forward(request,
        response);
    }
}
```

上述几个程序的关系如图 12.25 所示。

12.5.3　针对留言内容 Admin

在本节中，将针对 Admin 这个 POJO 类利用 DAO 模式实现对自己的 CRUD 操作。底层类 Admin.java 跟原来的一样，不需要进行修改。只需要新建两个类：AdminDAO.java 和 AdminDAOJdbc.java，代码 12.22 用来实现对 Admin 进行 CRUD 操作的接口类，代码 12.23 是实现前面接口的类。

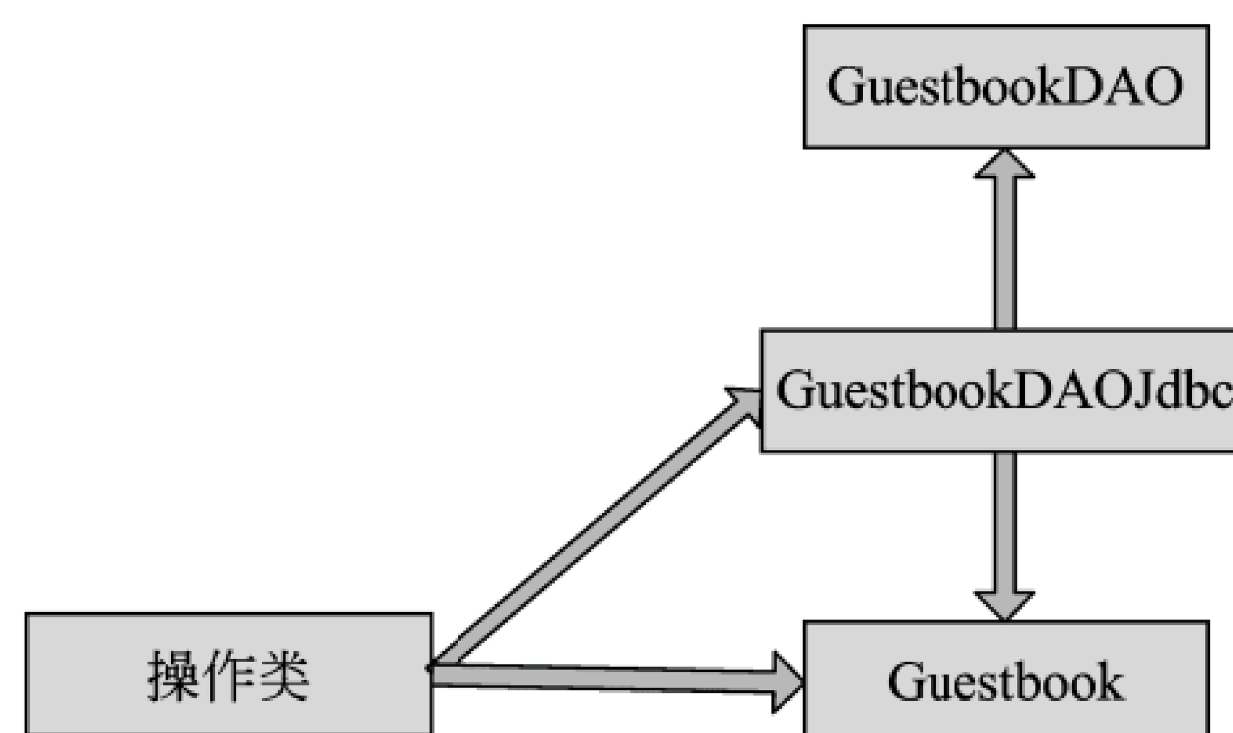


图 12.25　程序关系

代码 12.22　CRUD 的接口类：AdminDAO.java

```
...
public interface AdminDAO {
    public Admin getAdmin(String username, String password);
    //获取管理员对象
}
```

代码 12.23　CRUD 接口的实现类：AdminDAOJdbc.java

```
...
public class AdminDAOJdbc implements AdminDAO {
    //定义查找特定管理员对象
    private static final String SELECT_ADMIN_SQL = "select * from admin where
    username=? and password=?";
    OracleTool db = null;                              //初始化数据库工具类
    public AdminDAOJdbc() {                            //构造方法
        db = new OracleTool("java:/comp/env/jdbc/oracleds");
                                                       //对数据库工具赋值
```

```
        db.init();                                   //初始化工具类
    }
    public Admin getAdmin(String username, String password) {
        String params[] = { username, password };
                                                     //获取参数存储到 params 数组
        List list = (List) db.query(SELECT_ADMIN_SQL, params, new BeanList
        Handler(Admin.class));                       //调用工具类的 query()方法
        if (list.size() >= 1) {                      //判断变量 list
            return (Admin) list.get(0);
        } else {
            return null;
        }
    }
}
```

最后，修改 AdminUserServlet 操作类，代码 12.24 调用前面开发的两个类来检验该浏览者是否为管理权限。

代码 12.24　检验权限：AdminUserServlet.java

```
...
public class AdminUserServlet extends HttpServlet {
    private static final long serialVersionUID = 5801558969966197290L;
    //编写登录方法
    public void login(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        String message = "";                               //创建变量
        //获取相关参数
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        if (StringUtil.validateNull(username)) {           //判断参数 username
            message = "对不起，姓名不能为空，请您重新输入！<br>";
        } else if (StringUtil.validateNull(password)) {
                                                           //判断参数 password
            message = "对不起，密码不能为空，请您重新输入！<br>";
        } else {
            Admin tempUser = new Admin();                  //创建 Admin 对象
            tempUser.setUsername(username);
            tempUser.setPassword(password);
            AdminDAO dao= new AdminDAOJdbc();              //创建 AdminDAO 对象
            if (dao.getAdmin(username, password) == null) {
                message = "对不起，用户名或者密码错误";
                request.setAttribute("guesbook.admin.login.message",
                message);
                request.getRequestDispatcher("/admin/loginFail.jsp").
                forward(request, response);
            } else {
                HttpSession session = request.getSession();
                session.setAttribute("guesbook.admin.username",
                username);
                //实现转向 response.sendRedirect(request.getContext
                Path()+"/admin/secure/manage?q=list");
            }
        }
    }
    //编写退出方法
    public void logout(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
```

```
        HttpSession session = request.getSession();    //获取 Session 对象
        //移除 Session 对象
        session.removeAttribute("guesbook.admin.username");
        //实现转向

    response.sendRedirect(request.getContextPath()+"/admin/login.jsp");
    }
    //编写 doGet()方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        String method = request.getParameter("q"); //获取变量 q 的值
        if (method != null && method.equals("login")) {
                                                    //判断变量 method 的值
            login(request, response);
        } else {
            logout(request, response);
        }
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        doGet(request, response);                   //调用 doGet()方法
    }
}
```

上述几个程序文件的关系如图 12.26 所示。

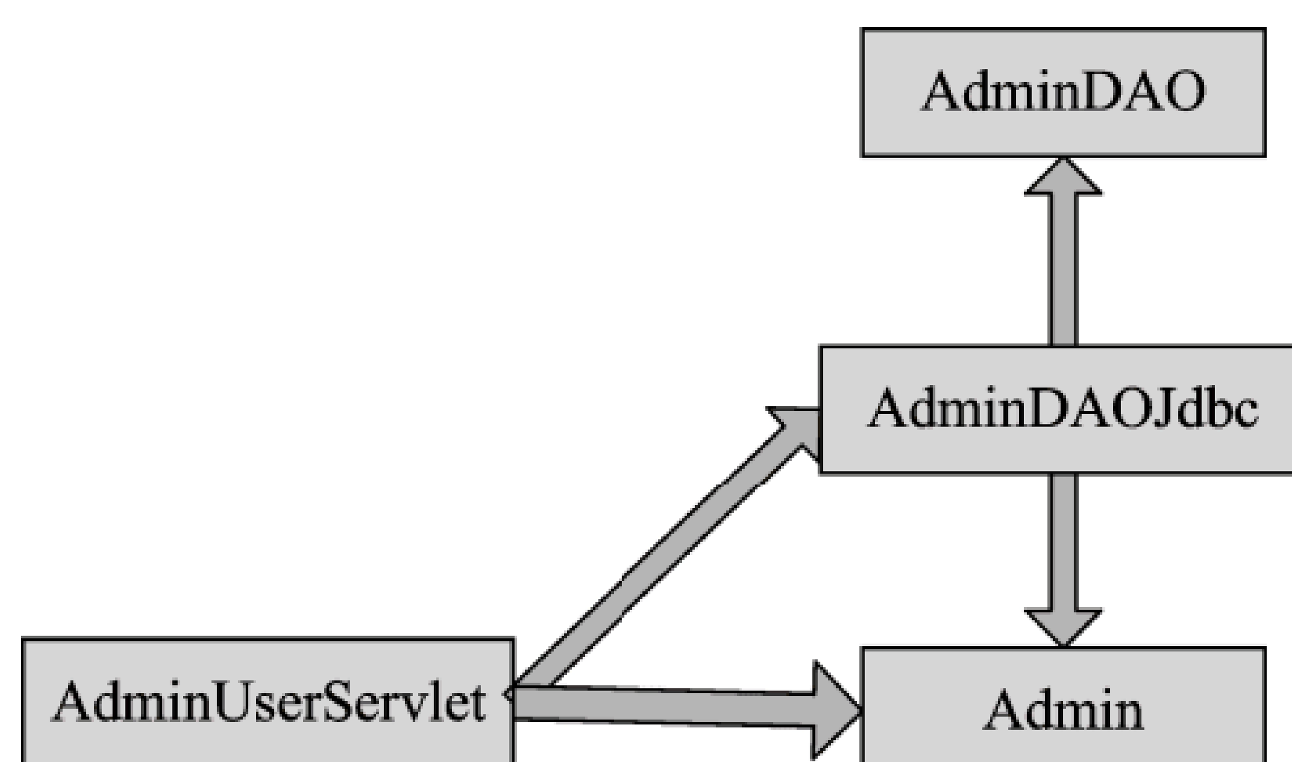


图 12.26　程序关系

12.6　小　　结

本章主要介绍网络留言板模块，该模块基于 JSP+Servlet+JavaBean 解决方案，保持了良好的 Java EE 中的 MVC 分层思想。

网络留言板模块在业务上主要分成 3 部分：添加留言功能、浏览留言功能和管理留言功能。在前两个部分中，主要利用两种方式 forward()和 sendRedirect()方法来实现页面的转向。而在最后一部分，不仅详细介绍了如何实现用户登录，而且还通过过滤器来实现页面的过滤。最后，本章还通过 DAO 模式来规范网络留言板代码。

第 13 章　网络留言板续——Oracle 数据库

在网络留言板模块中，当浏览者留言时，这些留言内容会存储在数据库中；当浏览者查看留言时，会从数据库中读取留言内容。对于每个大型网络系统来说，操作数据库是一个非常重要的模块，该模块性能的高低直接影响整个系统的性能。

本章将通过网络留言板的简单版本来讲解如何实现连接数据库、如何提高操作数据库的性能等。

13.1　连接数据库——JDBC 驱动程序

通过第 2 章内容可以知道 JDBC 程序分为：Java 到数据库协议、Java 到本地 API、JDBC-ODBC 桥和 Java 到网络协议，关于 JDBC 程序的工作原理如图 13.1 所示。

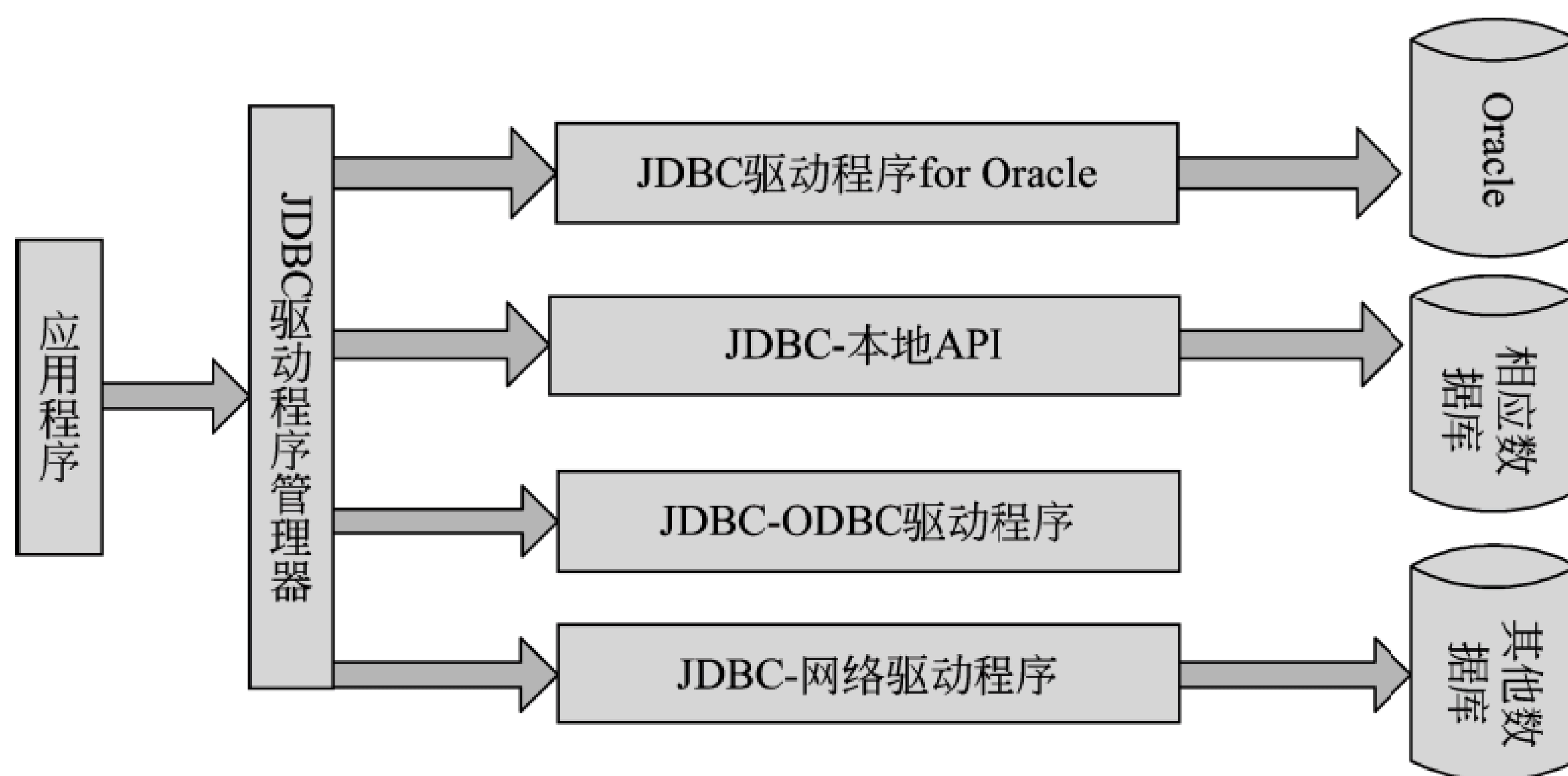


图 13.1　JDBC 的工作原理

13.1.1　利用 Java 到数据库协议方式连接数据库

当利用 Java 到数据库协议方式来操作数据库时，会利用数据库相关的协议把对驱动程序的请求直接发送给数据库，该协议实际上就是包含在驱动程序中的纯 Java 类。

本节将演示如何利用 oracle:thin 子协议来操作 Oracle 数据库，具体步骤如下。

（1）配置开发环境，引入与 Oracle 数据库相对应的 JDBC。首先复制对应的驱动程序，然后在 Package Explorer 视图中右击 jdbc 项目，在弹出的选项中选择 Paste 选项，就完成环境的配置。

（2）连接和操作 Oracle 数据库，代码 13.1 通过 oracle:thin 子协议显示出 orcl 数据库中

的 employee 表格中的数据。

代码 13.1　数据库协议方式：JDBCTest.java

```
...
    public static void main(String[] args) {
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            //加载 JDBC 驱动
            //连接数据库
            Connection conn = DriverManager.getConnection("jdbc:oracle:
            thin:@localhost:1521:orcl","scott","root");
            Statement stmt = conn.createStatement();        //获取陈述对象
            //获取结果集
            ResultSet rs = stmt.executeQuery("select * from employee");
            System.out.println("记录内容：");
            System.out.println("\tID 号\t 姓名\t 电话号码" );
            //遍历结果集
            while(rs.next()){
                System.out.print("\t" + rs.getInt(1));
                System.out.print("\t" + rs.getString(2));
                System.out.println("\t" + rs.getInt(3));
                System.out.println();
            }
            rs.close();
            stmt.close();
            conn.close();
        }catch(Exception e){
           e.printStackTrace();
        }
    }
}
```

【代码解析】

当程序通过 DriverManager.getConnection()方法连接 Oracle 数据库时，其参数 jdbc:oracle:thin:@localhost:1521:orcl 中使用的子协议名为 oracle:thin，子名称中 localhost 为主机名、1521 为 Oracle 服务器端口号和 orcl 为数据库名。

（3）编译和运行该程序后，其运行结果如图 13.2 所示。

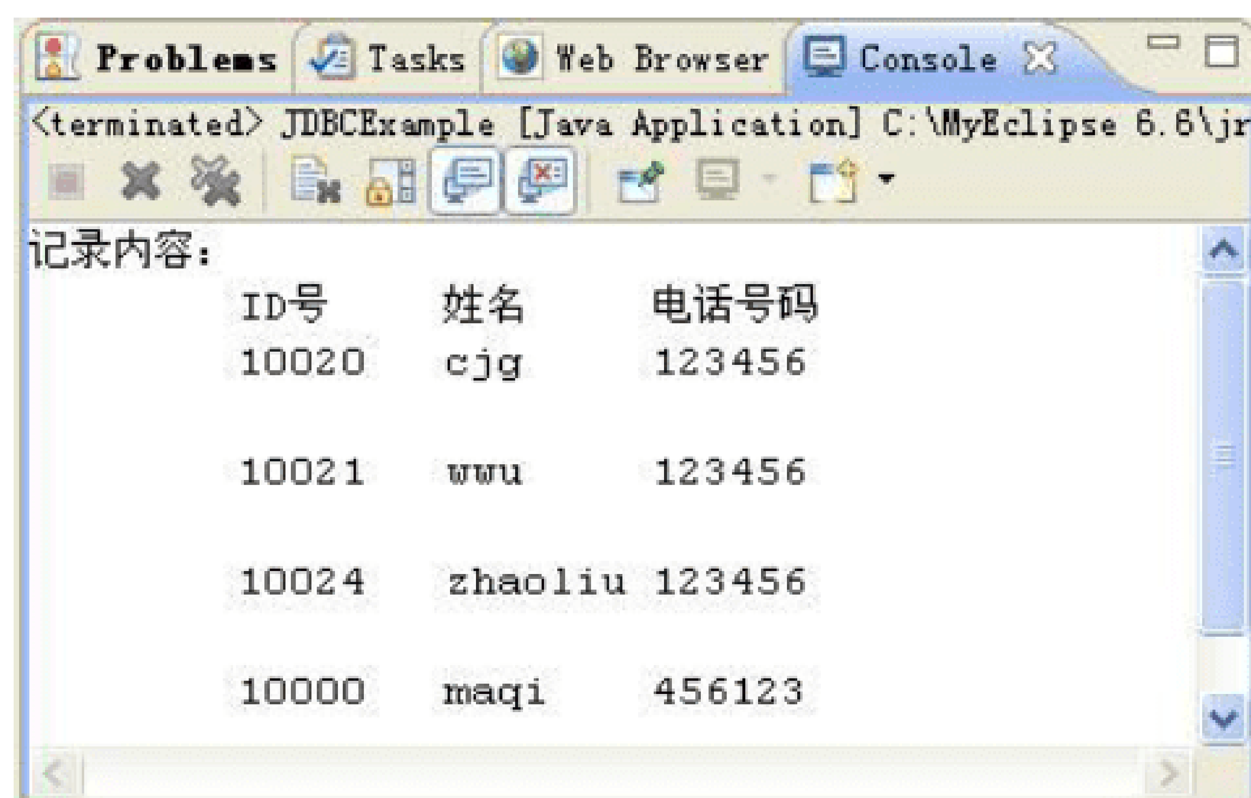


图 13.2　运行结果

13.1.2　利用 Java 到本地 API 方式连接数据库

当利用 Java 到本地 API 方式来操作数据库时，首先会通过调用本地的 API（数据库客户端提供的 API）连接到相应数据库客户端，然后再通过该客户端连接到相对应的数据库。在具体编写代码时，只需把调用驱动程序的代码转换成调用本地 API 的代码就可以。

本节将演示如何利用 OCI（Oracle Call Interface）子协议来操作 Oracle 数据库，所谓 OCI 方式就是应用程序先连接 Oracle 数据库的客户端，然后再通过该客户端连接数据库，因此该方式属于 Java 到本地 API 方式。具体步骤如下。

（1）Oracle 客户端配置，该功能可以通过 3 种方式来实现：利用客户端工具 Net Configuration Assistant；利用 Net Manager 图形化工具或修改 tnsnames.ora 数据库配置文件。

下面将通过 Net Configuration Assistant 工具来实现本地 Net 服务名配置，首先打开 Net Configuration Assistant 工具，在该工具的欢迎界面（如图 13.3 所示）中选择“本地 Net 服务名配置”单选按钮。单击“下一步”按钮后，就会进入“服务名配置”对话框（如图 13.4 所示）。在该对话框中选中“添加”单选按钮后，单击“下一步”按钮就会进入“服务名配置，服务名”对话框（如图 13.5 所示），在该对话框中需要为服务名选项填写安装 Oracle 服务器时产生的 SID（系统标示号）。然后单击“下一步”按钮进入“服务名配置，请选择协议”对话框（如图 13.6 所示）。在该对话框保持默认情况下单击“下一步”按钮后，就会进入“服务名配置，TCP/IP 协议”对话框（如图 13.7 所示）。在该对话框中需要对主机和端口号进行设置，该项目因为在本地计算机上，所以填写 127.0.0.1 端口号保持默认就可以。至此，基本完成 Oracle 客户端配置，对于其他的对话框保持默认就可以。



图 13.3　欢迎界面



图 13.4　服务名配置



图 13.5　添加服务名



图 13.6　选择协议



图 13.7　配置协议

（2）连接和操作 Oracle 数据库，代码 13.2 通过 OCI 子协议显示出 orcl 数据库中的 guestbook 表格中的数据。

代码 13.2　OCI 方式：OCITest.java

```
...
public class OCITest{
    public static void main(String args[]){
         try{
              Class.forName("oracle.jdbc.driver.OracleDriver");
              //加载驱动
              //连接数据库
             Connection conn = DriverManager.getConnection("jdbc:oracle:oci:
              @orcl","scott","root");
              Statement stmt = conn.createStatement();        //获取陈述对象
              //获取运行结果
              ResultSet rs = stmt.executeQuery("select * from guestbook");
              System.out.println("记录内容：");
              System.out.println("\tID 号\t 姓名\t 电话号码" );
              while(rs.next()){
                   System.out.print("\t" + rs.getInt(1));
                   System.out.print("\t" + rs.getString(2));
                   System.out.println("\t" + rs.getInt(3));
                   System.out.println();
              }
              rs.close();
              stmt.close();
              conn.close();
         }catch(Exception e){
              e.printStackTrace();
         }
    }
}
```

【代码解析】

当程序通过 DriverManager.getConnection()方法连接 Oracle 数据库时，其参数“jdbc:oracle:oci:@orcl”中使用的子协议名为 oracle:oci，子名称中@后面必须为刚才配置好的本地 Net 服务名。

（3）该程序如果想运行成功，必须要加入与 Oracle 数据库相对应的 JDBC 驱动程序。编译和运行该程序后，其运行结果如图 13.8 所示。

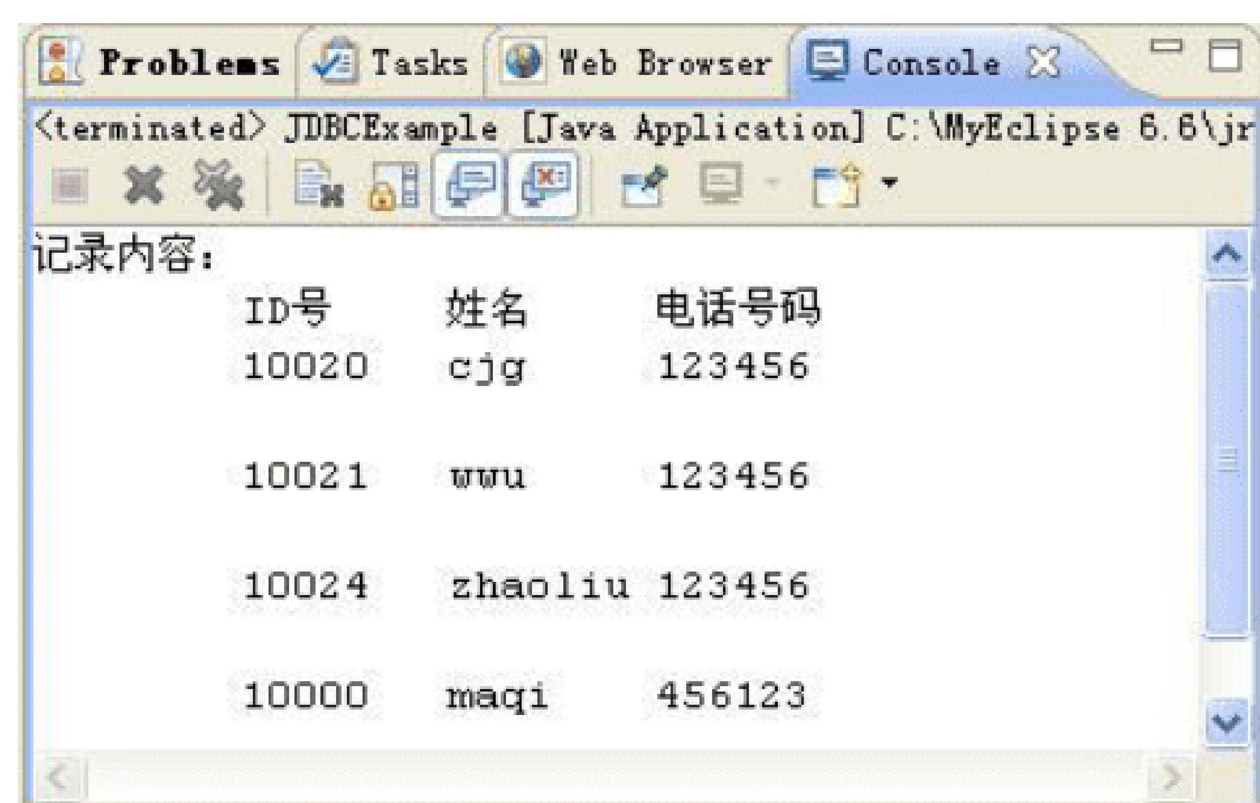


图 13.8　运行结果

13.1.3　利用 JDBC-ODBC 方式连接数据库

当利用 JDBC-ODBC 桥驱动程序来操作数据库时，该驱动程序并不是直接操作数据库驱动程序，而是调用 JDBC-ODBC 桥驱动程序操作 ODBC 驱动程序，进而连接各种数据库。其详细过程：应用程序中调用 JDBC-ODBC 桥驱动程序的指令，在“驱动管理器”的中转下交由 JDBC-ODBC 桥驱动程序处理，经过处理后这些指令格式就会转成 ODBC 的指令格

式。这些指令会自动通过 ODBC 的驱动程序，利用底层相应的数据库驱动程序来访问对应数据库，具体过程如图 13.9 所示。

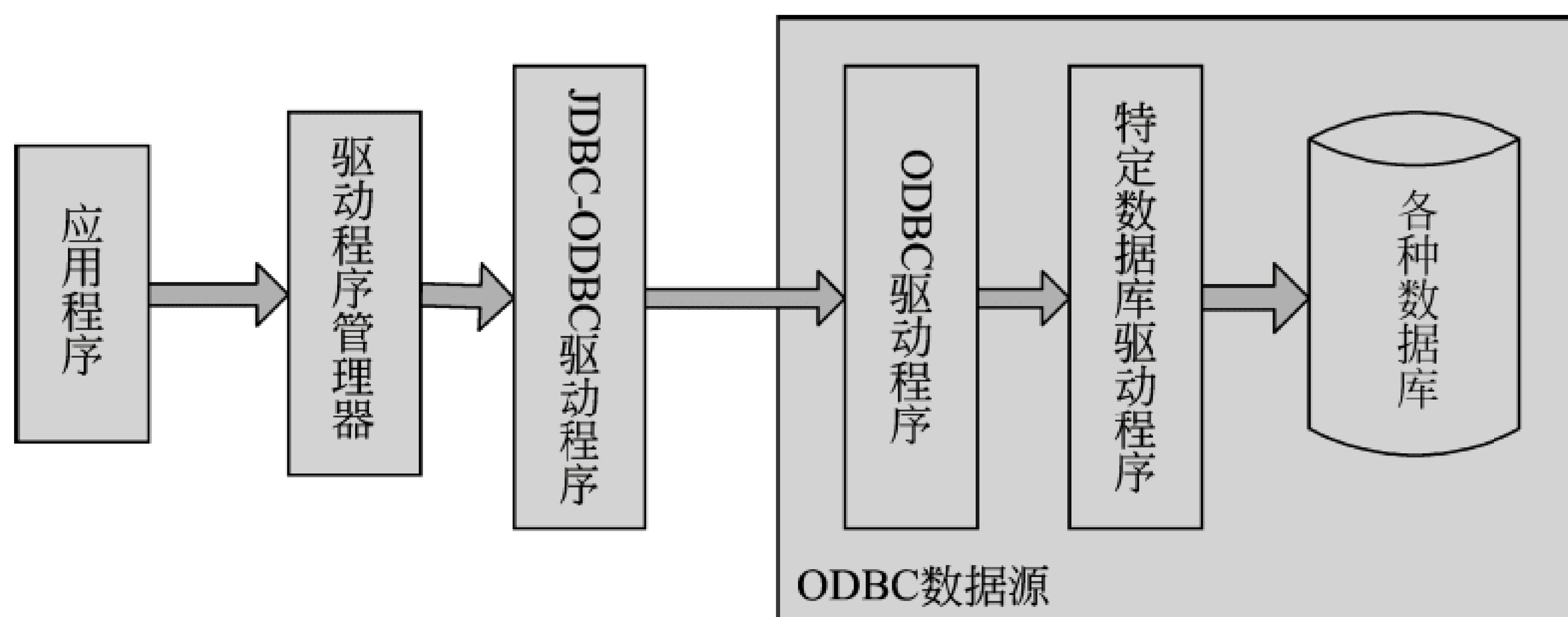


图 13.9　JDBC-ODBC 桥驱动程序使用

注意：对于开发人员来说，图 13.9 中最后三步是由 Windows 系统自动封装成为 ODBC 数据源，不需要进行相应的编程。

下面将演示如何利用 JDBC-ODBC 桥驱动程序操作数据库，具体步骤如下。

（1）创建 ODBC 数据源，微软的 Windows 系统全方位支持 ODBC 数据源。首先通过双击管理工具中的“数据源（ODBC）”图标打开 ODBC 数据源管理器（如图 13.10 所示）。然后在用户 DSN 标签中单击“添加”按钮，打开“创建数据源”对话框（如图 13.11 所示）。在该对话框中为操作的数据源选择相对应的驱动程序，因为该项目操作 Excel，所以选择“Driver do Microsoft Excel(*.xls)”选项。接着单击“完成”按钮就会出现关于 ODBC Microsoft Excel 安装的对话框，在该对话框中除了要自己定义一个数据源名外，还需要通过“选择工作簿”来设置数据源 Excel 的路径，设置结果如图 13.12 所示。最后单击“确定”按钮就会返回 ODBC 数据源管理器对话框（如图 13.13 所示），在该对话框中多出了一个名为 ExcelTest 的数据源记录。至此，ODBC 数据源就创建成功。

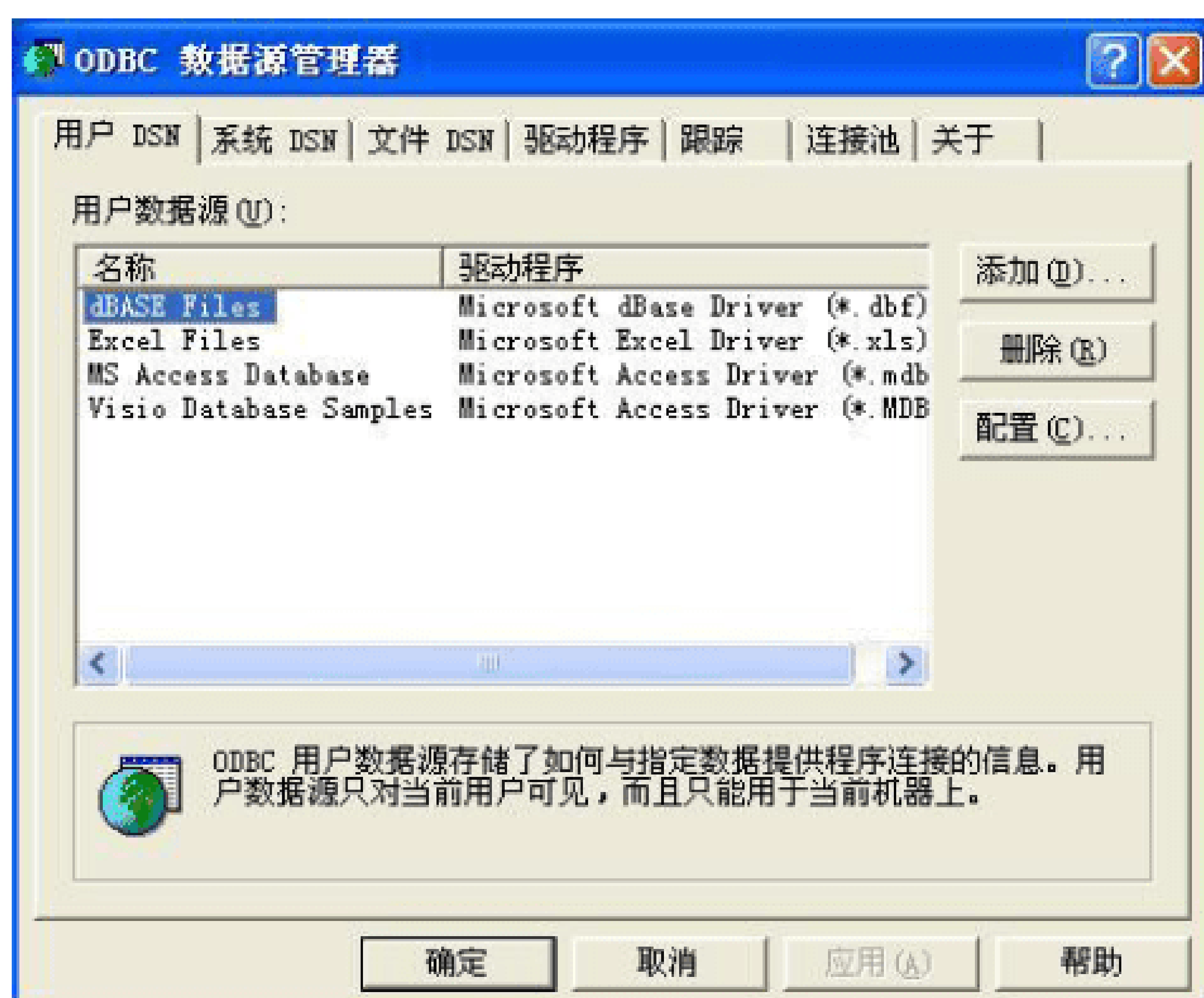


图 13.10　ODBC 数据源管理器



图 13.11　创建数据源

（2）连接和操作 ODBC 数据源，代码 13.3 通过 JDBC-ODBC 桥驱动程序来显示 Excel 表格中的数据。



图 13.12　ODBC Microsoft Excel 安装

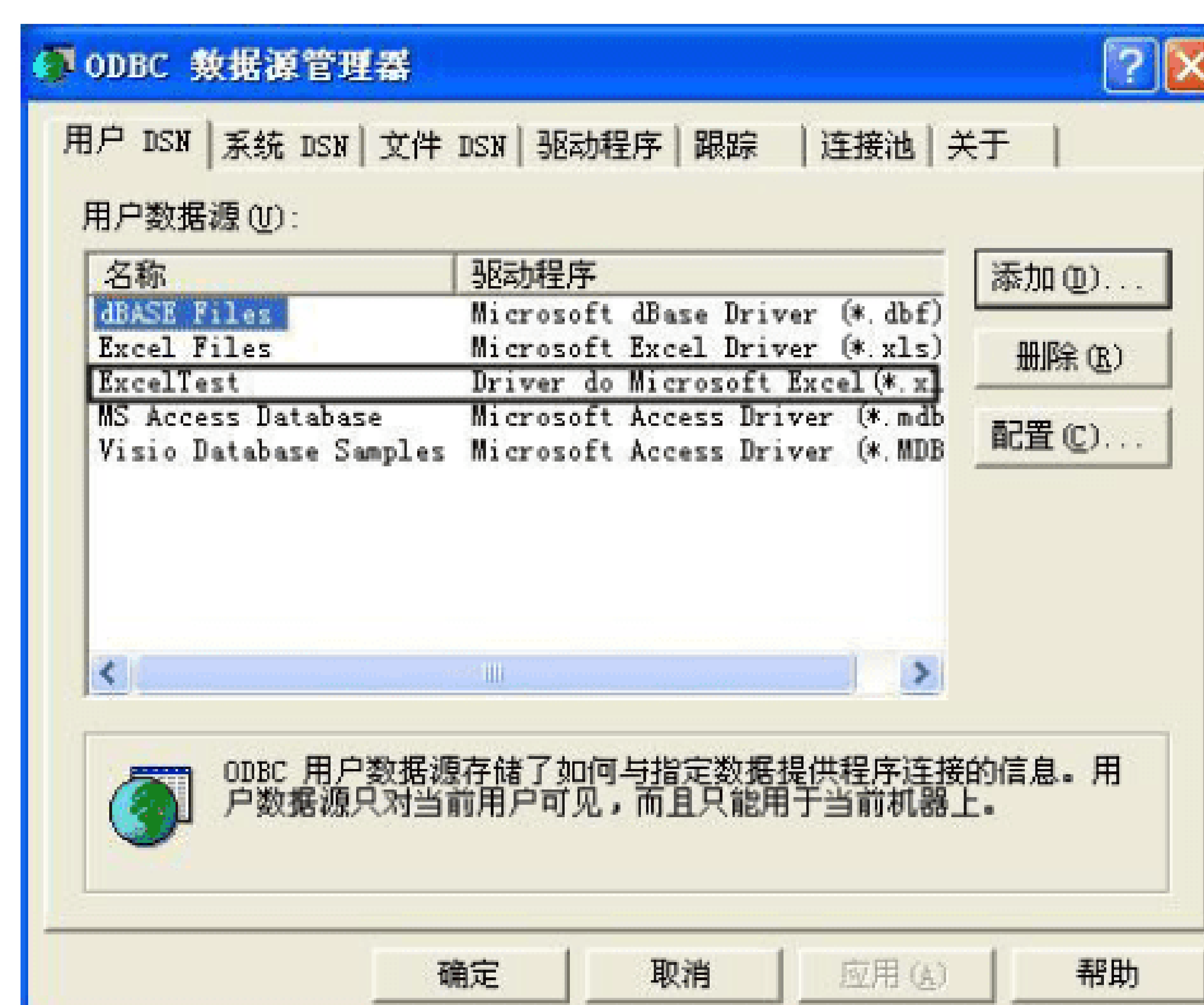


图 13.13　ODBC 数据源管理器

代码 13.3　操作 Excel 表格：JDBC_ODBCTest.java

```
...
import java.sql.*;
public class JDBC_ODBCTest{
    public static void main(String args[]){
    String drv = "sun.jdbc.odbc.JdbcOdbcDriver";       //创建驱动连接字符串
        try{
            Class.forName(drv);                         //加载驱动
            //获取连接
            Connection con = DriverManager.getConnection("jdbc:odbc:
            ExcelTest", "", "");
            Statement stmt=con.createStatement();       //获取陈述对象
            ResultSet rs=stmt.executeQuery("select * from [Sheet1$]");
            System.out.println("记录内容：");
            System.out.println("\tID 号\t 姓名\t 电话号码" );
            while(rs.next()){
                System.out.print("\t" + rs.getInt(1));
                System.out.print("\t" + rs.getString(2));
                System.out.print("\t" + rs.getInt(3));
                System.out.println();
            }
            rs.close();
            stmt.close();
            con.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

【代码解析】

- 当程序通过 Class.forName()方法加载驱动程序类时，其参数必须为 sun.jdbc.odbc.JdbcOdbcDriver，即 JDBC-ODBC 桥驱动程序。
- 当程序通过 DriverManager.getConnection()方法连接 ODBC 数据源时，其第 1 个参数中的协议与子协议名必须为 jdbc 和 odbc，而子名称是确定的必须为 ExcelTest 数据源名。第 2 个和第 3 个参数用来设置用户名和密码，当它们为空时可以设置为空字符串。

🔔 **注意**：在早期由于经常会使用 JDBC-ODBC 桥驱动程序操作数据库，所以 Sun 公司在 JDK 中就提供了 JDBC-ODBC 桥驱动程序，因此在程序中不需要加入 JDBC-ODBC 桥驱动程的架包。

（3）编译和运行该程序后，其运行结果如图 13.14 所示。

Problems　Tasks　Web Browser　Console
<terminated> JDBC_ODBCTest (1) [Java Application] C:\MyEclipse 6.6\jre\bin
记录内容：

ID号	姓名	电话号码
1	cjg	123456
2	wwu	123456
3	zliu	456123

图 13.14　运行结果

13.2　数据库连接池

当编写对于数据库的访问不是很频繁应用程序时，可以在需要访问数据库时创建一个新连接，用完后就关闭它，这样做不会带来什么明显的性能上的开销。但是对于一个复杂的数据库应用，情况就完全不同了，频繁地建立、关闭连接，会极大地降低该应用程序的性能。

13.2.1　数据库连接池简介

为了避免频繁地建立和关闭数据库，开发人员可以通过建立一个数据库连接池和一套管理策略来达到连接资源的共享，使得对于数据库的连接可以是高效、安全的。

对于共享资源，有一个很著名的设计模式：资源池。该模式正是为了解决资源频繁分配、释放所造成的问题。把该模式应用到数据库连接管理领域，就是建立一个数据库连接池，提供一套高效的连接分配、使用策略，最终目标是实现连接的高效、安全的复用。

数据库连接池的基本原理是在内部对象池中，维护一定数量的数据库连接，并对外暴露数据库连接获取和返回方法。如图 13.15 所示为当程序中需要建立数据库连接时，只需从内存中获取一个数据库连接来用，而不是新建一个数据库连接，在使用完毕后，只需放回内存即可。对于连接的建立和断开都由连接池自己管理。

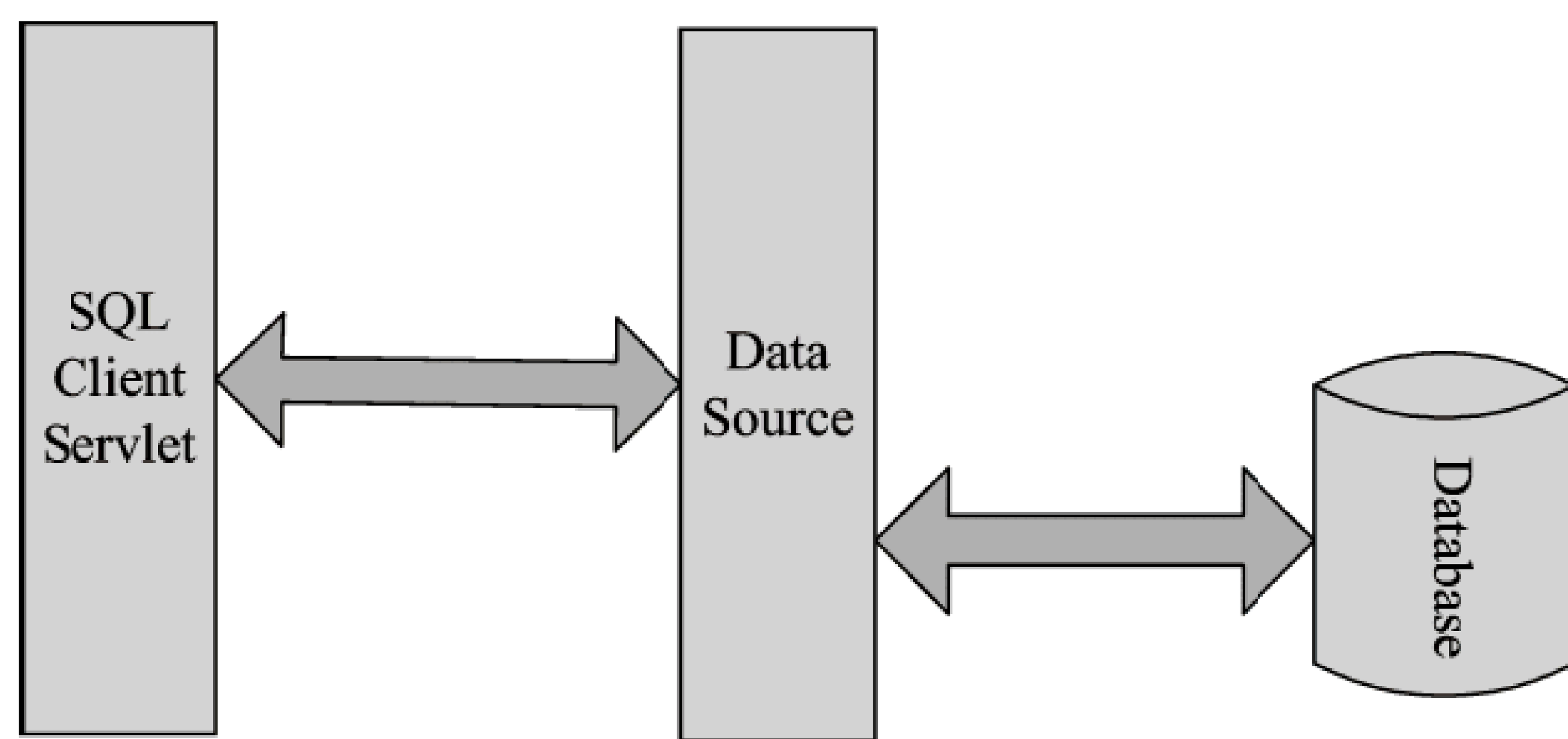


图 13.15　数据库连接池原理

综上所述，使用数据库连接池具有如下特点。

- 资源重用：为了避免频繁地创建、释放数据库连接，实现了数据库连接的重用。
- 高效的系统响应：数据库连接池在初始化过程中，一般就会创建若干个数据库连接存储在池中备用，所以具有高效的效率。
- 统一的连接管理：对于连接数目的创建、断开、管理和关闭等操作都是由数据库连接池统一管理。

13.2.2　数据库连接池原理

通过 13.2.1 节可以知道数据库连接池的简单概念，本节将通过一个简单的实例来演示如何实现数据库连接池和使用数据库连接池后的效果，在该实例中连接池是通过 JDBC 驱动程序来实现连接。具体步骤如下。

（1）首先创建一个名为 connectionpooling 的 Java 项目。

（2）接着创建两个 class 文件：实现数据库连接池程序 ConnectionPool.java 和测试连接池程序 ConnectionPoolTest.java。代码 13.4 用来实现数据库连接池，代码 13.5 用来显示使用数据库连接池与不使用连接池的性能的差别。

代码 13.4　数据库连接池：ConnectionPoolTool.java

```
...
public class ConnectionPool {
    private Vector<Connection> pool;
    private String url;
    private String username;
    private String password;
    private String driverClassName;
    private int poolSize = 1;   //连接池的大小，也就是连接池中有多少个数据库连接
    private static ConnectionPool instance = null;
    //私有的构造方法，禁止外部创建本类的对象，要想获得本类的对象，通过<code>
    getIstance</code>方法
    private ConnectionPool() {
        init();
    }
    //连接池初始化方法，读取属性文件的内容，建立连接池中的初始连接
    private void init() {
        pool = new Vector<Connection>(poolSize);
        readConfig();
        addConnection();
    }
    //返回连接到连接池中
    public synchronized void release(Connection conn) {
        pool.add(conn);
    }
    //关闭连接池中的所有数据库连接
    public synchronized void closePool() {
        for (int i = 0; i < pool.size(); i++) {
            try {
                ((Connection) pool.get(i)).close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            pool.remove(i);
        }
    }
    //返回当前连接池的一个对象
    public static ConnectionPool getInstance() {
        if (instance == null) {
            instance = new ConnectionPool();
        }
        return instance;
    }
```

```
    //返回连接池中的一个数据库连接
    public synchronized Connection getConnection() {
        if (pool.size() > 0) {
            Connection conn = pool.get(0);
            pool.remove(conn);
            return conn;
        } else {
            return null;
        }
    }
    //在连接池中创建初始设置数据库连接
    private void addConnection() {
        Connection conn = null;
        for (int i = 0; i < poolSize; i++) {
            try {
                Class.forName(driverClassName);
                conn = java.sql.DriverManager.getConnection(url, username,
                password);
                pool.add(conn);
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
    //读取设置连接池的属性文件
    private void readConfig() {
        try {
            String path = System.getProperty("user.dir") + "\\dbpool.
            properties";
            FileInputStream is = new FileInputStream(path);
            Properties props = new Properties();
            props.load(is);
            this.driverClassName = props.getProperty("driverClassName");
            this.username = props.getProperty("username");
            this.password = props.getProperty("password");
            this.url = props.getProperty("url");
            this.poolSize = Integer.parseInt(props.getProperty("pool
            Size"));
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("读取属性文件出错. ");
        }
    }
}
```

【代码解析】

在具体开发程序时并不需要程序员自己实现数据库连接池，因为许多公司或组织已经把性能更好的数据库连接池组件集成到自己的软件中，程序员在需要使用数据库连接池时只需要简单配置一下就可以。

代码 13.5　数据库连接池性能：ConnectionPoolTest.java

```
...
public class ConnectionPoolTest {
    public static void main(String[] args) throws Exception {
        String sql = " select * from student ";              //定义 SQL 语句
```

```
        long start = System.currentTimeMillis();   //定义一个时间变量
        ConnectionPool pool = null;                //定义一个数据库连接池变量
        for (int i = 0; i < 100; i++) {
            pool = ConnectionPool.getInstance();   //初始化数据库连接池
            Connection conn = pool.getConnection();    //获取连接对象
            Statement stmt = conn.createStatement();   //创建陈述对象
            ResultSet rs = stmt.executeQuery(sql);     //执行 SQL 语句
            while (rs.next()) {
            }
            rs.close();                                //关闭数据集
            stmt.close();                              //关闭陈述对象
            pool.release(conn);                        //返回连接对象
        }
        pool.closePool();                              //关闭数据库连接池
        System.out.println("经过 100 次的循环调用，使用连接池花费的时间:" +
        (System.currentTimeMillis() - start) + "ms\n");
        String hostName = "127.0.0.1";                 //安装数据库的计算机
        String driverClass = " com.mysql.jdbc.Driver";
                                               //加载 MySql 数据库驱动程序
        String url = " jdbc:mysql://hostname:3306/testmysql";
                                               //数据库 URL
        String user = "root";                  //用户名
        String password = "root";              //用户密码
        start = System.currentTimeMillis();
        for (int i = 0; i < 100; i++) {
            Class.forName(driverClass);        //加载数据库驱动
            Connection conn = DriverManager.getConnection(url, user,
            password);
            Statement stmt = conn.createStatement();       //创建陈述对象
            ResultSet rs = stmt.executeQuery(sql);         //执行 SQL 语句
            while (rs.next()) {
            }
            rs.close();
            stmt.close();
            conn.close();
        }
        System.out.println("经过 100 次的循环调用，不使用连接池花费的时间:" +
        (System.currentTimeMillis() - start) + "ms");
    }
}
```

【代码解析】

在上述代码中，第 1 个 for 循环通过数据库连接池实现 100 次数据库的连接和关闭，而第 2 个 for 循环中通过直接调用 JDBC 驱动程序来实现 100 次数据库的连接和关闭。

（3）在 connectionpooling 项目中创建一个名为 dbpool.properties 的属性文件，代码 13.6 用来配置数据库连接池连接数据库的信息。

代码 13.6　连接数据库信息：dbpool.properties

```
driverClassName=oracle.jdbc.driver.OracleDriver    //加载数据库驱动程序
username=scott                                 //用户名
password=root                                  //用户密码
url=jdbc:oracle:thin:@localhost:1521:orcl1     //数据库 URL
poolSize=10                                    //数据库连接数目
```

（4）编译和运行 ConnectionPoolTest 程序后，其运行结果如图 13.16 所示。

注意： 虽然测试程序的运行结果，会根据具体的计算机性能而显示的时间不一样，但是不使用数据库连接池花费的时间，基本上是使用数据库连接池的 n 倍。

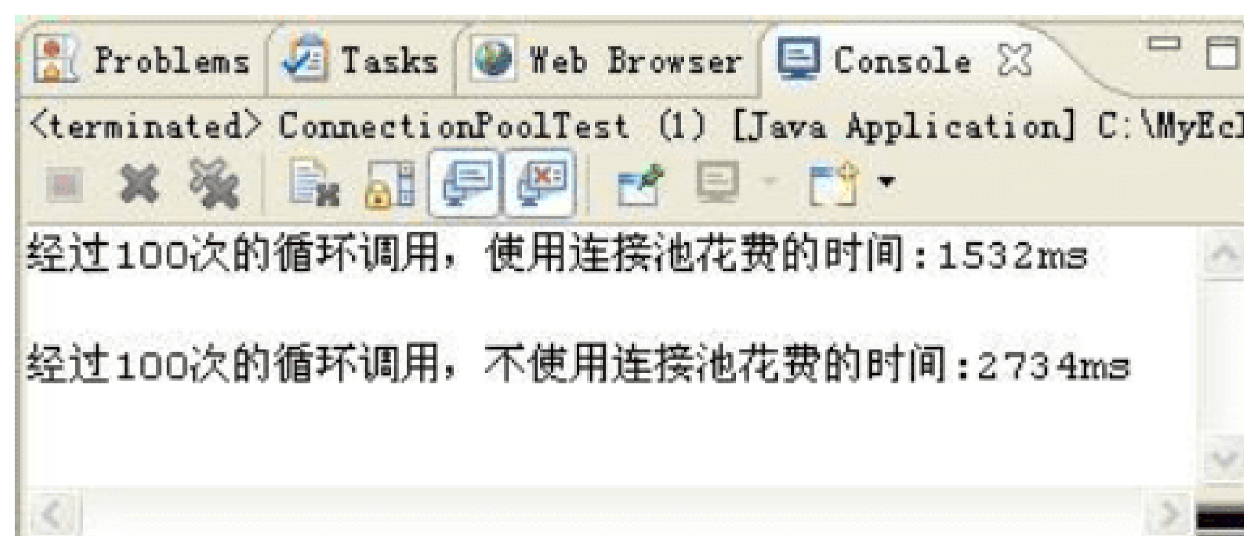


图 13.16　运行结果

13.2.3　配置和使用服务器 Tomcat 连接池

在开发具体项目时编写数据库连接池是没有必要的，因为现在已经存在许多数据库连接池的现成组件，只需要配置一下就可以使用。而且现在许多应用服务器都已经内置了数据库连接池，如 Tomcat 服务器、Jboss 服务器和 WebLogic 服务器等。

为了便于讲解，根据网络留言板实现其的一个精简版本，来演示如何利用数据库连接池操作数据库。具体步骤如下。

（1）打开文件夹 Tomcat 6.0 根目录\conf 中的文件 context.xml，代码 13.7 中对配置文件代码的修改使得 Tomcat 支持数据库连接池。

代码 13.7　修改配置文件：context.xml

```
<!--属性 reloadable 表示应用发生变化，随时都可以侦测-->
<Context  reloadable="true" >
    <!--侦测 WEB-INF/web.xml 内容是否更改-->
<WatchedResource>WEB-INF/web.xml</WatchedResource>
    <!-- Uncomment this to disable session persistence across Tomcat restarts
     -->
    <!--
    <Manager pathname="" />
    -->
    <!-- Uncomment this to enable Comet connection tacking (provides events
        on session expiration as well as webapp lifecycle) -->
    <!--
    <Valve className="org.apache.catalina.valves.CometConnectionManager
    Valve" />
    -->
<!--Resource 设置数据库连接池的核心-->
        <Resource name="jdbc/oracleds"
        auth="Container"
        type="javax.sql.DataSource"
        maxActive="100" maxIdle="30"maxWait="10000"
        username="scott" password="root"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        url="jdbc:oracle:thin:@localhost:1521:orcl"
/>
</Context>
```

【代码解析】

在元素<Resource>中，通过属性 name 来设置该数据源的名称；属性 auth 用来设置验证方式；属性 type 用来设置资源类型；属性 driverClassName 用来设置数据库驱动；属性 url 用来设置数据库 URL。

（2）接着创建一个名为 simplemessagebook 的项目，由于该项目是网络留言板 messagebook 的精简版，所以只详细地讲解涉及数据库池的内容。代码 13.8 和代码 13.9 分别为实现添加留言内容和显示留言内容的功能。

代码 13.8　添加留言内容：AddMessageServlet.java

```
...
public class AddMessageServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //定义 SQL 语句变量
        String sql = "insert into guestbook (id,name,email,phone,title,
        content,time) values(gb seq.nextval,?,?,?,?,?,?)";
        //创建各种变量
        int result = 0;
        Connection conn = null;
...
            try {
                /*
                * 在下面的字符串"java:comp/env/jdbc/ oracleds"中，*"java:
                comp/env/"是不变的，
                * 而"jdbc/ oracleds "Tomcat 服务器中配置文件数据源名称
                */
                Context context = new InitialContext();    //上下文对象
                DataSource ds = (DataSource) context.lookup("java:/comp/
                env/jdbc/oracleds");                        //数据源对象
                conn = ds.getConnection();                  //连接数据库
                PreparedStatement pstmt = conn.prepareStatement(sql);
                                                            //创建陈述对象
                //对陈述对象中各个占位符进行设置
                pstmt.setString(1, StringTool.filterHtml(name));
                pstmt.setString(2, StringTool.filterHtml(request.get
                Parameter("email")));
                pstmt.setString(3, StringTool.filterHtml(request.get
                Parameter("phone")));
                pstmt.setString(4, StringTool.filterHtml(title));
                pstmt.setString(5, request.getParameter("content"));
                SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
                hh:mm:ss");
                pstmt.setString(6, sdf.format(new java.util.Date()));
                result = pstmt.executeUpdate();
                                          //执行 prepareStatement 对象
                pstmt.close();
            } catch (NamingException e) {
                e.printStackTrace();
            } catch (SQLException e) {
                e.printStackTrace();
            } finally {
                try {
                    conn.close();              //关闭连接
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
...
}
```

【代码解析】

- ❑ 类 Context 是上下文环境，即容器 Tomcat 的整体环境。如果想了解该类可以查看关于 JNDI 的相关资料。具体实现是通过 InitialContext()初始化上下文环境得到上下文环境变量 context。
- ❑ 方法 context.lookup()用来获取上下文，在该方法的参数 java:/comp/env/jdbc/oracleds 中，前部分 java:/comp/env/表示环境命名上下文，一般不会改变；后部分 jdbc/oracleds 表示配置文件中名为 jdbc/oracleds 的对象。具体意义就是从相应配置文件中加载名为 jdbc/oracleds 的对象，然后返回数据源对象 ds。
- ❑ 通过 ds.getConnection()方法获取数据库连接 conn 后，接着再通过 prepareStatement 类把 SQL 语句传送给数据库返回结果。在具体编写 prepareStatement 方面的代码时通过两个步骤来实现返回结果，即首先通过 conn. prepareStatement()方法创建 PreparedStatement 语对象，传入该方法中 SQL 语句参数一般会带有占位符。接着通过 setXXX 方法设置占位符参数。最后通过 executeUpdate()方法执行 PreparedStatement 语句返回结果，如果 SQL 语句执行成功返回 1，否则为 0。

代码 13.9　显示留言内容：GetMessagesServlet.java

```
...
public class GetMessagesServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        String sql = "select * from guestbook order by id desc";
                                                        //创建 SQL 语句变量
        Connection conn = null;                         //创建连接对象
...
        try {
            //获取数据源对象
            Context initContext = new InitialContext();
            DataSource ds = (DataSource) initContext.lookup("java:/comp/
            env/jdbc/oracleds");
            conn = ds.getConnection();                  //连接数据库
            PreparedStatement pstmt = conn.prepareStatement(sql);
                                                        //获取陈述对象
            ResultSet rs = pstmt.executeQuery();
                                                //执行 prepareStatement 对象
            //遍历结果集
            while (rs.next()) {
                this.printRow(out, rs);
            }
            rs.close();
            pstmt.close();
        } catch (NamingException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
...
}
```

注意： 在上述代码中最后通过 executeQuery()方法执行 PreparedStatement 对象，该方法的返回结果是 ResultSet 类对象。

（3）最后通过配置 web.xml 文件来设置数据源，代码 13.10 用来实现对数据源的配置。

代码 13.10　配置文件：web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
...
  <resource-ref>
<description>Oracle Datasource </description>          <!-- 描述信息 -->
<res-ref-name>jdbc/oracleds</res-ref-name>              <!-- 数据源名字 -->
<res-type>javax.sql.DataSource</res-type>               <!-- 数据源的类型 -->
    <res-auth>Container</res-auth>                      <!-- 设置验证方式 -->
  </resource-ref>
</web-app>
```

（4）单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/simplemessagebook/addMessage.htm，运行结果如图 13.17 所示。



图 13.17　运行结果

13.3　Commons DbUtils 组件

对于程序员来说，编写关于 JDBC 方面的编码是一个重复性非常高的工作。虽然好的 JDBC 代码并不难编写，但是一不小心就会产生一些低级的错误，于是就需要使用 Commons DbUtils 组件。虽然 Commons DbUtils 组件是一个非常简单的组件，但是其却使得关于 JDBC 代码的编写变得非常容易。

13.3.1　下载 Commons DbUtils 组件

Commons DbUtils 组件是一个精密而简单的组件，实际上该组件只是封装了一些常用

的 JDBC 操作。目前最稳定的版本为 1.1 版本，具体下载步骤如下。

（1）首先访问下载 Commons DbUtils 组件的官方网站（http://jakarta.apache.org/），如图 13.18 所示。



图 13.18　下载首页

（2）打开 Commons DbUtils 组件官方网站的首页后，在其左边的 Ex-Jakarta 目录中选择 Commons 选项，就会转到如图 13.19 所示的 Commons 组件列表页面。

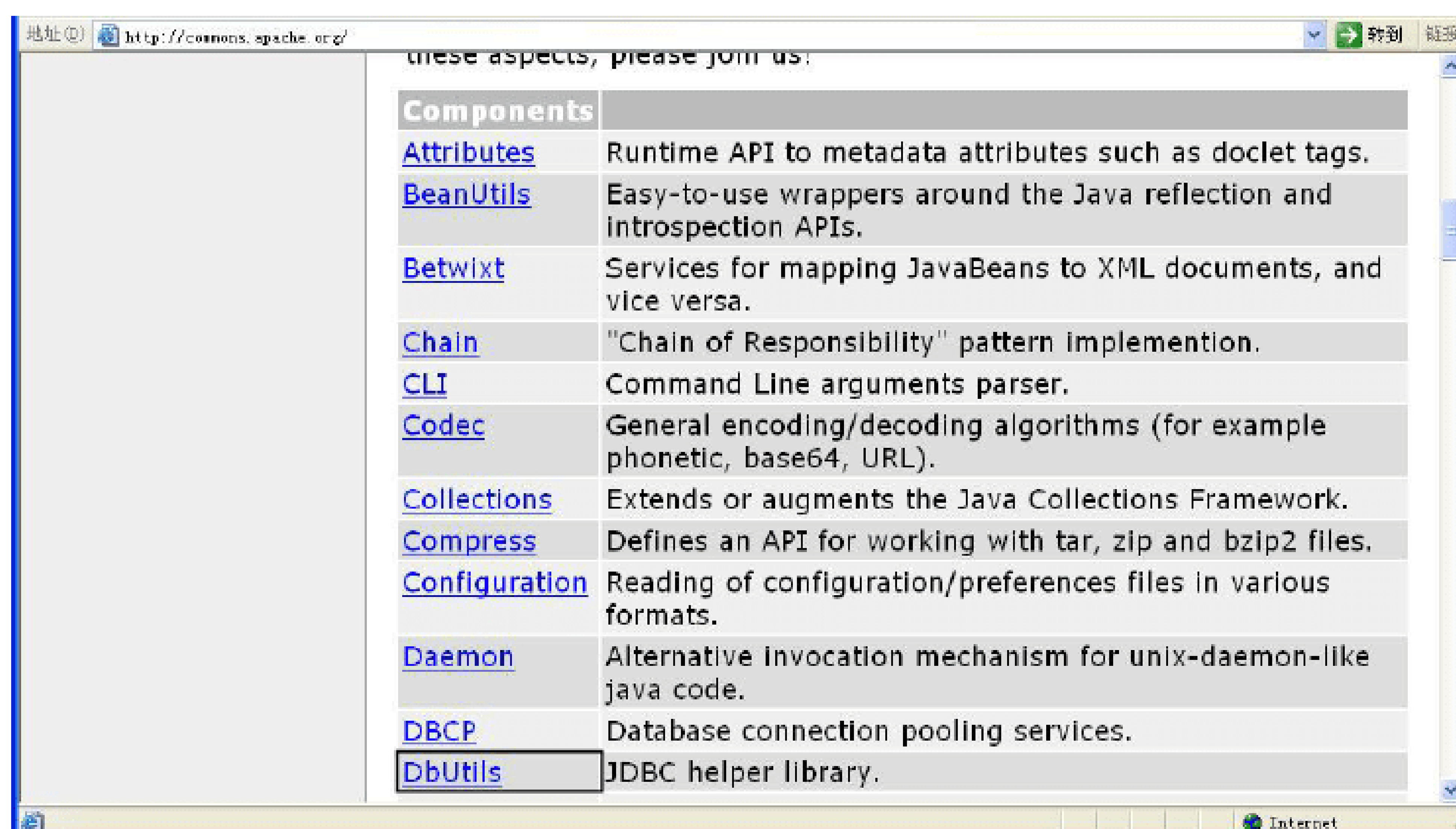


图 13.19　Commons 组件列表页面

（3）在 Commons 组件列表页面中，单击 DbUtils 链接后，就可以转到如图 13.20 所示的关于 Commons DbUtils 架包页面。



图 13.20　Commons DbUtils 架包页面

（4）在关于 Commons DbUtils 架包页面的左边目录中选择 Download 选项，就会转到如图 13.21 所示的关于 Commons DbUtils 架包下载页面，在该页面中选择 1.1.zip 选项就会实现该架包的下载。



图 13.21　下载 Commons DbUtils 架包

至此，就完成对 Commons DbUtils 架包的下载。

13.3.2　Commons DbUtils 架包使用

13.3.1 节介绍了如何下载 Commons DbUtils 架包，下载完该架包后就可以在 Java Web 项目中使用该框架。在具体使用之前，先解压 commons-dbutils-1.1.zip 文件，该压缩包目录如图 13.22 所示。

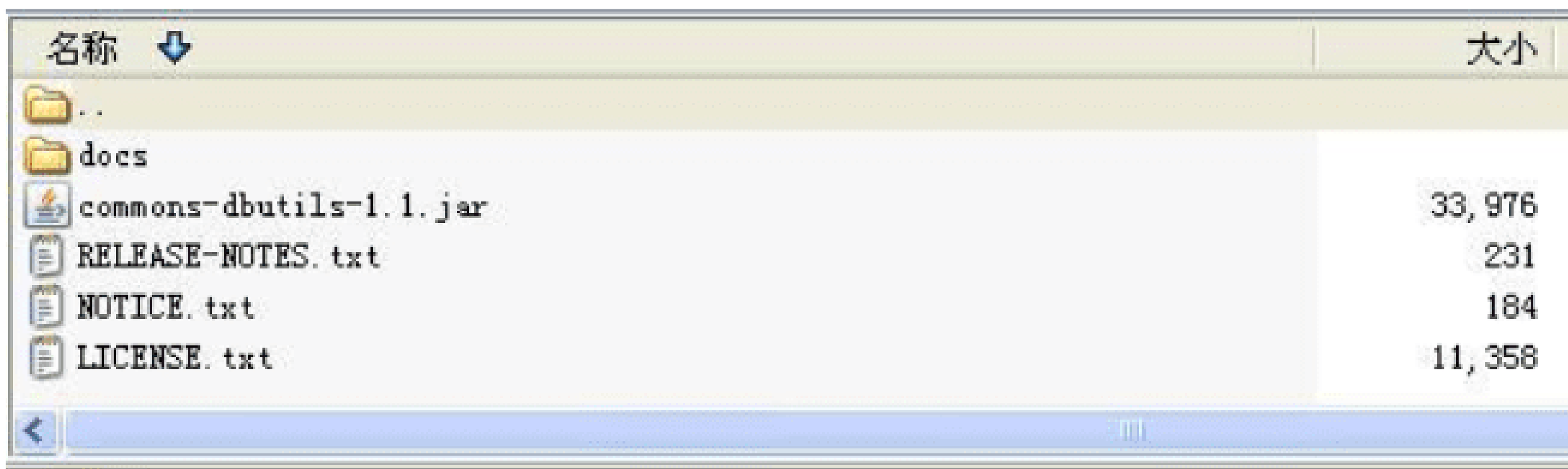


图 13.22　目录结构

在上述目录中存在两个主体文件，它们分别如下。

- commons-dbutils-1.1.jar：关于 Commons DbUtils 组件的类库。
- docs：关于 Commons DbUtils 组件的帮助文件。

Commons DbUtils 是一个非常小的组件，所以涉及的类也不多，其中最重要的类如下。

- DbUtils 类：Commons DbUtils 架包的启动类或开始类，用来设置所要连接的数据库、连接数据库的类型等，该类里所有的方法都是静态，具体方法如表 13.1 所示。

表 13.1　DbUtils类中的方法

方　法　名	作　　用
Close	该方法通过检查所提供的参数是不是为 NULL 来决定是否关闭连接等
CloseQuietly	用来关闭连接、声明和结果集
CommitAndCloseQuietly	用来提交连接，然后关闭连接
LoadDriver	用来装载并注册 JDBC 驱动程序

- ResultSetHandler 类：用来把 jaca.sql.ResultSet 结果集转换成符合程序员要求的类型。在该接口中提供了一个单独方法：Object handle (java.sql.ResultSet rs)，该方法的参数为结果集，而返回的类型为 Object。MapListHandler 类和 BeanListHandler

类都是实现该接口的类，其中在 MapListHandler 类中，把结果集存储到 List 集合中，每一个记录都是 Map 类型；在 BeanListHandler 类中，仍然把结果集存储到 List 集合中，但是每一个记录都是 JavaBean 类型。

- QreryRunner 类：用来执行 SQL 语句，其返回对象一般为 MapListHandler 类对象或 BeanListHandler 类对象，该类中重要方法如表 13.2 所示。

表 13.2　QreryRunner类中的方法

方　法　名	作　　用
Query()	用来执行查询操作
Update()	用来执行插入、更新或删除操作

为了便于讲解，根据网络留言板实现其的一个精简版本，来演示如何利用 Commons DbUtils 架包实现数据库操作。具体步骤如下。

（1）首先把 Commons DbUtils 架包中的 commons-dbutils-1.1.jar 类库引入项目 dbutilsmessagebook 中

（2）由于该项目是网络留言板 messagebook 的精简版，所以只详细地讲解涉及数据库的内容。代码 13.11 和代码 13.12 分别为实现添加留言内容和获取留言内容的功能。

代码 13.11　添加留言内容：AddMessageServlet.java

```
...
public class AddMessageServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //设置 SQL 语句
        String sql = "insert into guestbook (id,name,email,phone,title,
        content,time) values(gb_seq.nextval,?,?,?,?,?,?)";
        int result = 0;                                  //创建一个变量
...
        if (StringTool.validateNull(name)) {
        ......);
        } else if (StringTool.validateNull(title)) {
        ......);
        } else {
...
            try {
                Context initContext = new InitialContext();
                                                    //获取上下文对象
                //获取数据源
                DataSource ds = (DataSource) initContext.lookup("java:/
                comp/env/jdbc/oracleds");
                QueryRunner qr = new QueryRunner(ds);
                                                    //获取 QueryRunner 对象
                result = qr.update(sql, param);     //执行 SQL 语句
            } catch (NamingException e) {
                e.printStackTrace();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            //判断 result 值
            if (result == 0) {
                out.println("对不起，添加留言不成功，请您重新输入！<BR>");
                out.println("<a href=\"" + request.getContextPath() +
```

```
                    "/addMessage.htm\">添加新的留言</a><BR>");
                } else {
                    out.println("祝贺您，成功添加留言。<BR>");
                    out.println("<a href=\"" + request.getContextPath() +
                    "/servlet/getMessages\">查看所有留言内容</a><BR>");
                }
                out.println(" </body>");
                out.println("</html>");
                out.flush();
                out.close();
            }
    }
}
```

【代码解析】

首先获取数据源对象 ds，接着通过 QueryRunner 类的带参构造函数获取该类对象 qr，该构造函数中的参数为数据源 ds。接着通过调用 QueryRunner.update()方法来实现更新功能的 SQL 语句，该方法带有 2 个参数，其中第 1 个参数用来为 SQL 语句，第 2 个参数为对于 SQL 语句中的占位符参数。

代码 13.12　显示留言内容：GetMessagesServlet.java

```
...
public class GetMessagesServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        String sql = "select * from guestbook order by id desc";
                                                          //设置 SQL 语句
...
        try {
            Context initContext = new InitialContext();     //获取上下文对象
            //获取数据源
            DataSource ds = (DataSource) initContext.lookup("java:/comp/
            env/jdbc/oracleds");
            QueryRunner qr = new QueryRunner(ds);  //创建 QueryRunner 对象
            MapListHandler handler = new MapListHandler();
                                                    //创建 MapListHandler
            List list = (List) qr.query(sql, handler);      //执行 SQL 语句
            for (int i = 0; i < list.size(); i++) {         //遍历结果集
                Map map = (Map) list.get(i);
                printRow(out, map);
            }
        } catch (NamingException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        out.println("</center></body>");
        out.println("</html>");
        out.flush();
        out.close();
    }
...
}
```

【代码解析】

首先获取数据源对象 ds，接着通过 QueryRunner 类的带参构造函数获取该类对象 qr，该构造函数中的参数为数据源 ds。接着创建一个 MapListHandler 类对象，该类实现了 ResultSetHandler 接口。最后通过调用 QueryRunner.Query()方法来实现查询功能的 SQL 语句,该方法带有两个参数,其中第 1 个参数是 SQL 语句，第 2 个参数是 SQL 语句中的占位符参数，即把返回结果集存储到 List 集合中，每一个记录都是 Map 类型。

（3）单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080//dbutilsmessagebook/addMessage.htm，运行结果如图 13.23 所示。



图 13.23　运行结果

13.3.3　利用 Commons DbUtils 工具类操作数据库

在 13.3.2 节的 dbutilsmessagebook 项目中，关于数据库的操作代码分散到了各个页面中，当想阅读、调试、维护时存在许多不便。如果想解决这个弊端，可以设计一个关于数据库操作所有方法的类，当想操作数据库时，直接调用该类的方法则可以。具体步骤如下：

（1）首先在 messagebooktool 中创建一个 java 文件，代码 13.13 利用 Commons DbUtils 组件实现对数据库操作。

代码 13.13　操作数据库工具类：OracleTool.java

```
...
public class OracleTool {
    private String dataSourceName;                          //数据源名字变量
    private DataSource ds;                                  //数据源变量
    public OracleTool(String dataSourceName) {              //有参构造函数
        this.dataSourceName = dataSourceName;
    }
    public OracleTool() {                                   //无参构造函数
    }
    public void setDataSourceName(String dataSourceName) {
                                                    // 设置属性 dataSourceName
        this.dataSourceName = dataSourceName;
    }
    public void init() {                                    //初始化连接数据库
        Context initContext;
        try {
            initContext = new InitialContext();             //获取 Context 对象
            ds = (DataSource) initContext.lookup(dataSourceName);
                                                            //获取数据源
        } catch (NamingException e) {
            e.printStackTrace();
        }
    }
```

```
    public int update(String sql, String[] param) {     //实现对数据库的更新
        int result = 0;
        QueryRunner qr = new QueryRunner(ds);           //获取与数据库的连接
        try {
            result = qr.update(sql, param);             //获取更新结果
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return result;                                  //返回结果
    }
    public Object query(String sql, String[] param, ResultSetHandler rsh)
    { //实现对数据库查询
        QueryRunner qr = new QueryRunner(ds);           //获取与数据库的连接
        Object result = null;
        try {
            result = qr.query(sql, param, rsh);         //获取查询结果
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return result;                                  //返回结果
    }
}
```

【代码解析】

- 在 update()方法中，首先通过 QueryRunner 类的带参构造函数获得一个与数据库的连接。接着通过 QueryRunner 类中带有两个参数的 update()方法来实现更新功能，在调用 query()方法时，第一个参数为执行更新功能的 SQL 语句，可以实现记录的添加、修改和删除功能；第二个参数用来对应 SQL 语句中的“?”占位符。
- 在 query()方法中，首先通过 QueryRunner 类的带参构造函数获得一个与数据库的连接。接着通过 QueryRunner 类中带有 3 个参数的 query()方法来实现查询功能，在调用 query()方法时，第 1 个参数为执行查询功能的 SQL 语句，可以实现记录的添加、修改和删除功能；第 2 个参数用来对应 SQL 语句中的“?”占位符；第 3 个参数用来设置返回结果的类型，即根据实现 ResultSetHandler 接口的类设置从 ResultSet 得来的记录类型。最后要记得把结果通过 return 返回。

（2）由于该项目是网络留言板 messagebook 的精简版，所以只详细地讲解涉及调用数据库工具类的内容。代码 13.14 和代码 13.15 分别为实现添加留言内容和获取留言内容的内容。

代码 13.14　添加留言内容：AddMessageServlet.java

```
...
public class AddMessageServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        String sql = "insert into guestbook (id,name,email,phone,title,
        content,time) values(gb_seq.nextval,?,?,?,?,?,?)"; //设置 SQL 语句
        int result = 0;                                    //创建表示结果变量
...
        if (StringTool.validateNull(name)) {
...
        } else if (StringTool.validateNull(title)) {
...
```

```
        } else {
...
            //设置参数数组
            String param[] = { StringTool.filterHtml(name),
    StringTool.filterHtml(request.getParameter("email")),
            StringTool.filterHtml(request.getParameter("phone")), String
            Tool.filterHtml(title),
            request.getParameter("content"), sdf.format(new java.util.
            Date()) };
            //创建 OracleTool 对象
            OracleTool db = new OracleTool("java:/comp/env/jdbc/
            oracleds");
            db.init();                              //初始化 OracleTool 对象
            result = db.update(sql, param);         //执行 SQL 语句
            if (result == 0) {
...
            } else {
...
            }
        }
...
    }
}
```

【代码解析】

首先通过 OracleTool 类的带参数构造函数来创建 OracleTool 对象 db，该方法中的参数为数据源名字。接着通过调用 db.init()方法来初始化数据库连接。最后，通过调用 db.update()方法来获取结果集，如果 SQL 语句执行成功则返回 1；否则为 0。

代码 13.15　获取留言内容：GetMessagesServlet.java

```
...
public class GetMessagesServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        String sql = "select * from guestbook order by id desc";
                                                //设置 SQL 语句
        List list = null;                       //设置一个结果变量
        //创建 OracleTool 对象
        OracleTool db = new OracleTool("java:/comp/env/jdbc/oracleds");
        db.init();                              //初始化 OracleTool 对象
        //执行 SQL 语句
        list = (List) db.query(sql, null, new BeanListHandler(Message
        book.class));
...
    }
}
```

【代码解析】

在上述代码中当执行 query()方法时，该方法需要 3 个参数，第 1 个参数是 SQL 语句，第 2 个参数为 SQL 语句中的占位符参数，第 3 个参数用来设置返回的结果类型。最后结果存储到 List 数据集中，每个记录为 Messagebook 类型。

（3）单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/messagebooktool/addMessage.jsp，运行结果如图 13.24 所示。

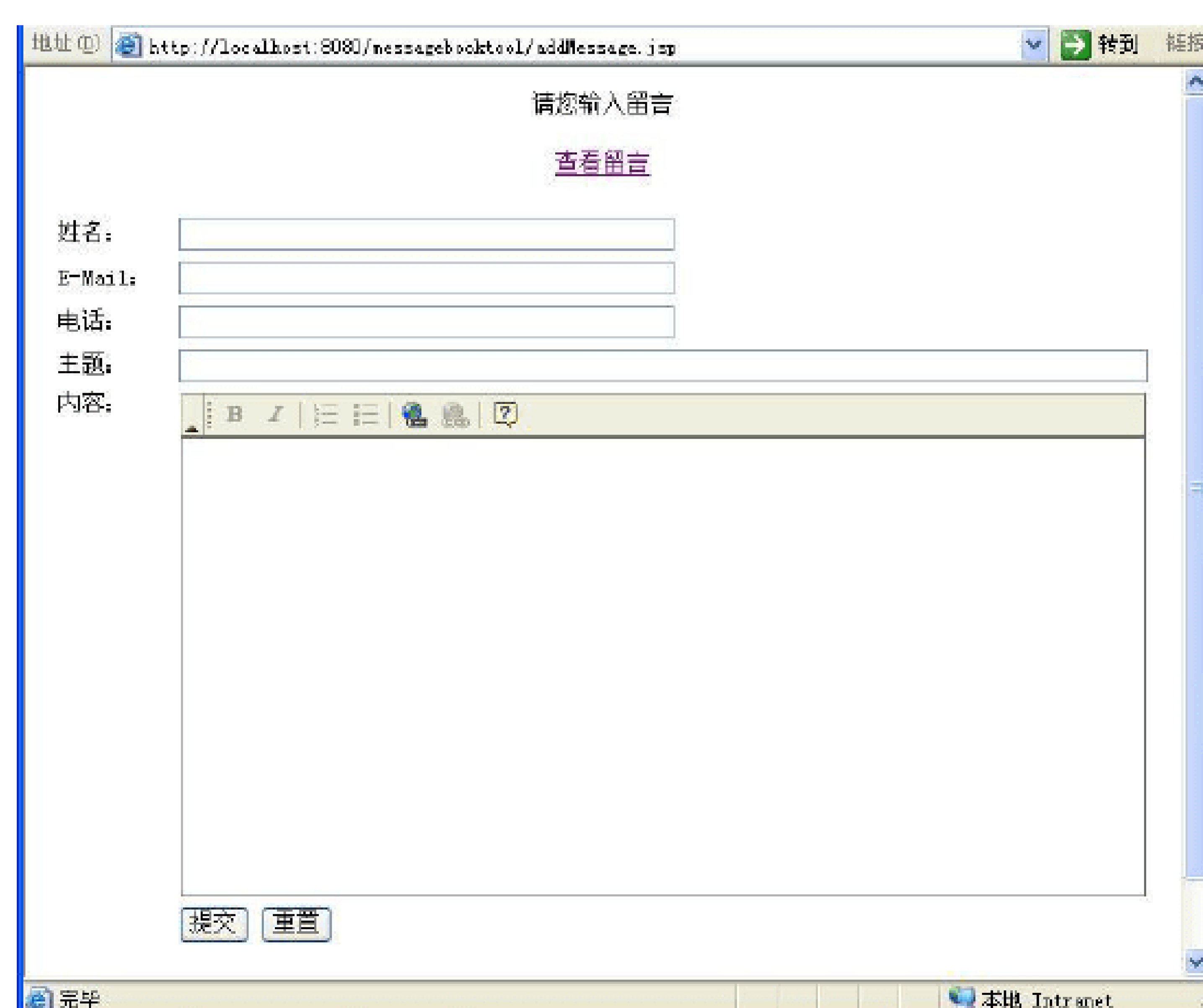


图 13.24　运行结果

13.4　小　　结

本章为第 12 章网络留言板项目的续章，主要讲解 Oracle 数据库的使用。为了能够讲清楚关于 Oracle 数据库的相关操作，本章首先讲解了使用 JDBC 驱动程序的 3 种方式：数据库协议方式连接数据库、本地 API 方式连接数据库和 JDBC-ODBC 方式连接数据库。接着通过配置 Tomcat 服务器的连接池，详细讲解了 Oracle 数据库的高级运用——数据库连接池。最后还讲解了封装 JDBC 驱动操作的 Commons DbUtils 组件。

第 14 章　AJAX 技术 JQuery 框架的经典应用

一个功能强大的网站系统为了能够方便用户的使用，一般会用到 AJAX 技术。对于一些比较小的应用，使用 XMLHTTPRequest 对象就可以实现。对于大的应用，如果还是使用 XMLHTTPRequest 对象来实现则会显得太复杂，这时就需要封装了关于 XMLHTTPRequest 对象操作的 AJAX 框架。

本章将详细介绍关于 AJAX 技术 JQuery 框架的详细内容，为了能够让读者掌握 JQuery 框架，本章实现了关于 JQuery 框架的经典案例。

14.1　JQuery 框架的简单应用

在 Java Web 项目中为了方便用户实现 AJAX 技术的应用，经常会使用 AJAX 的各种框架。AJAX 框架出现的目标就是使繁琐的 AJAX 开发变得更加容易，让程序员不用重复地构造 JavaScript 的底层。本章将介绍关于 AJAX 的一些简单基础知识，以及关于 JQuery 框架的简单应用。

14.1.1　关于 AJAX 框架

面对市场上众多的 AJAX 框架，如何选择适合自己项目的 AJAX 框架呢？这主要跟程序员要开发的项目有关，即在具体开发之前，需要弄清楚项目的如下问题：

- 是否需要支持各种类型的浏览器？
- 是否需要限制框架的大小？
- 是否需要提高代码的执行速度？
- 是否需要统一 JavaScript 和 HTML 代码的风格?

下面将简单介绍一下流行的 AJAX 框架。

1. Dojo框架

该框架主要用来解决 AJAX 技术易用性和特效问题的轻巧性框架，以降低网页或网页应用程式前端开发速度著称。

- 优点：支持拖拉、淡出和淡入、移动、透明、操作 SVG 图档等动态效果，同时也支持分页标签（tab）、树状结构、日历、文字编辑器等特殊效果。最重要的是该功能还支持使用「上一页」与加入「我的最爱」功能。
- 缺点：由于该框架内容特别复杂，技术文档极端不全，各个版本间的改动比较大，所以掌握起来不容易。

2. DWR框架

该框架全称 Direct Web Remoting，利用其可以实现客户端 JavaScript 和服务器端 Java 的相互交互。

- ❑ 优点：DWR 框架最大的好处就在于让 Java 开发人员可以利用熟悉的语法来处理页面与资料，并且能配合 Struts、Tapestry 来使用。
- ❑ 缺点：当客户端呼叫服务器的 Java 程式时，会存在一些安全上的隐患。

3. JQuery框架

该框架以 Prototype 框架为基础，简化并精炼了 JavaScript 语法。

- ❑ 优点：把简化 JavaScript 语法做到了一种极端，最重要的在于强大的存取页面元素功能，无论是文件的节点、CSS 的选取或 XPath 表达式，都能利用「$（ ）」函式快速存取，并赋予它们更多的功能。
- ❑ 缺点：由于 JQuery 框架简化了 JavaScript 语言的语法，所以程序的内容就发展成另一种样子，需要重新掌握该语法。

关于 AJAX 的框架还有很多，本节只讲这些，感兴趣的读者可以查阅相关资料。由于本章只使用 JQuery 框架，所以关于其他框架在其他章节中将不再出现。

14.1.2　JQuery 框架的下载和配置

JQuery 框架由 John Resig 于 2006 年初创建，主要用来简化 JavaScript 和 AJAX 的编程。使用该框架可以非常方便地在网页上实现操作文档、处理事件、实现特效并为 Web 页面添加 AJAX 方面的交互。

目前 JQuery 框架比较稳定的版本为 1.3.2，具体的下载步骤如下。

（1）首先访问下载 JQuery 组件的官方网站（http://jquery.com），如图 14.1 所示。在该页面中单击 Download 链接就可以转到关于 JQuery 组件的相关页面。

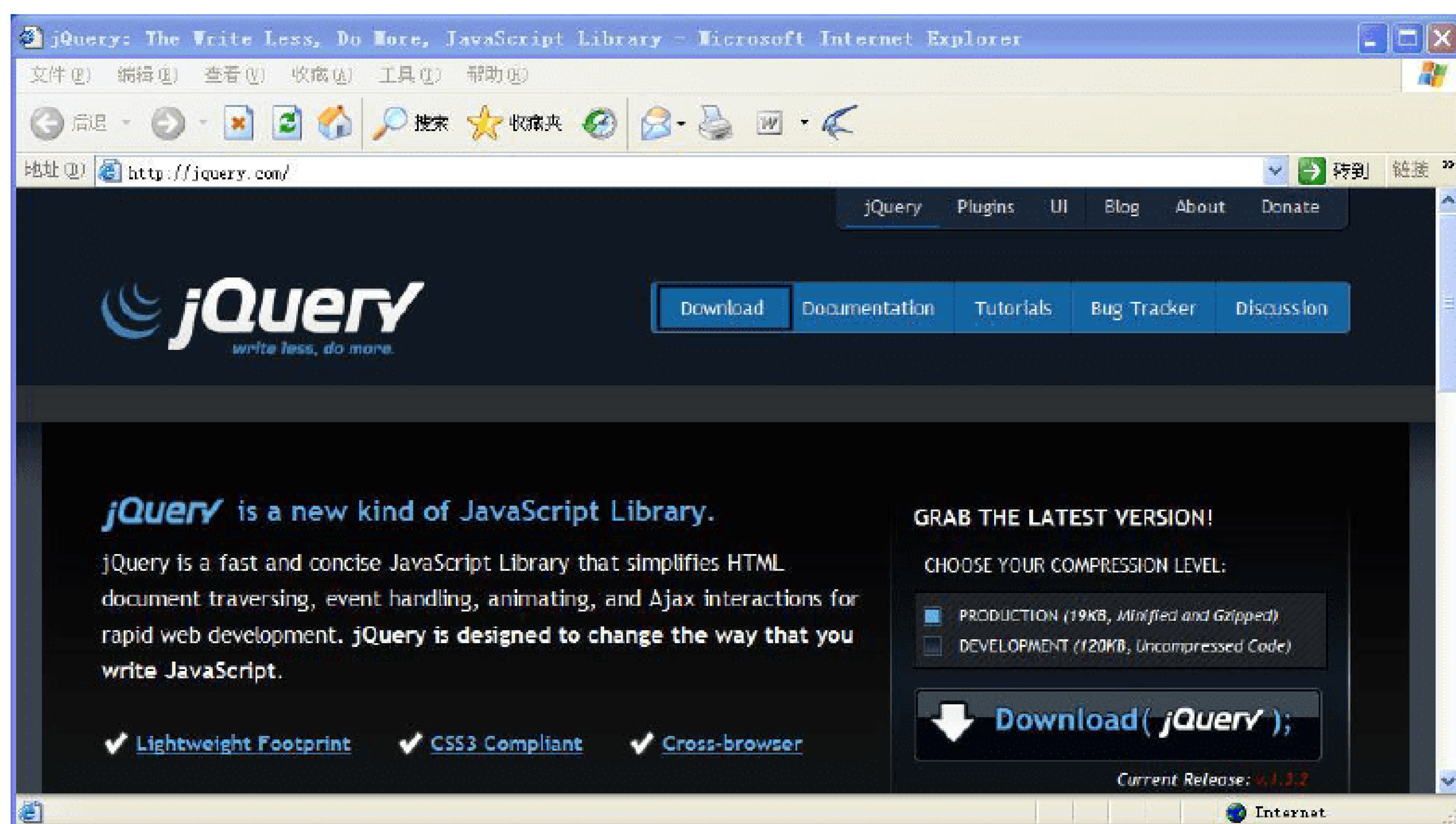


图 14.1　JQuery 组件首页

（2）在 JQury 组件相关页面中（如图 14.2 所示），单击 Uncompressed 链接就可以转

到关于该组件下载的页面。

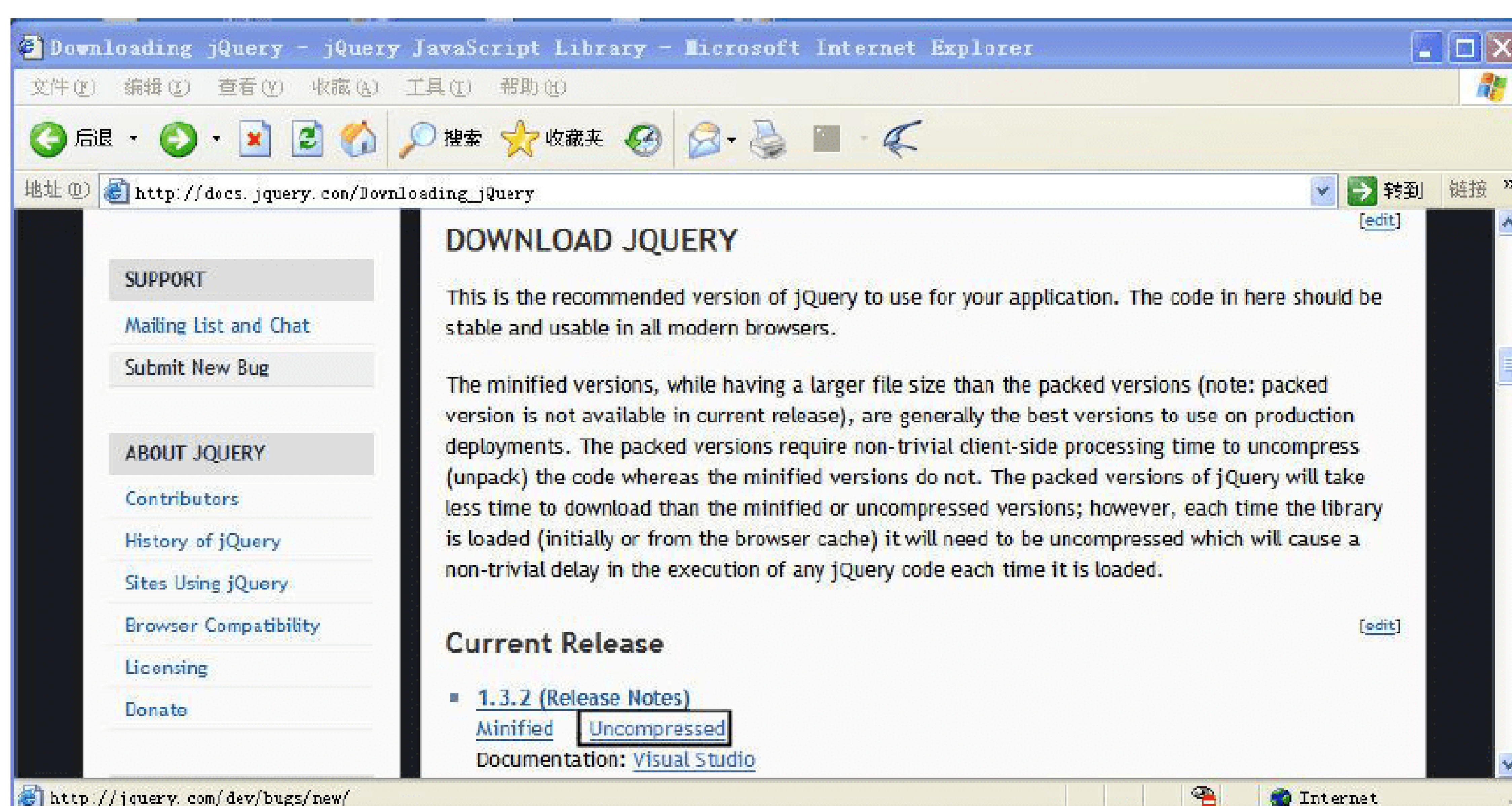


图 14.2　JQuery 组件相关页面

（3）在关于 JQuery 组件下载的相关页面中（如图 14.3 所示），单击 jquery-1.3.2.js 链接就可以实现该版本组件的下载。

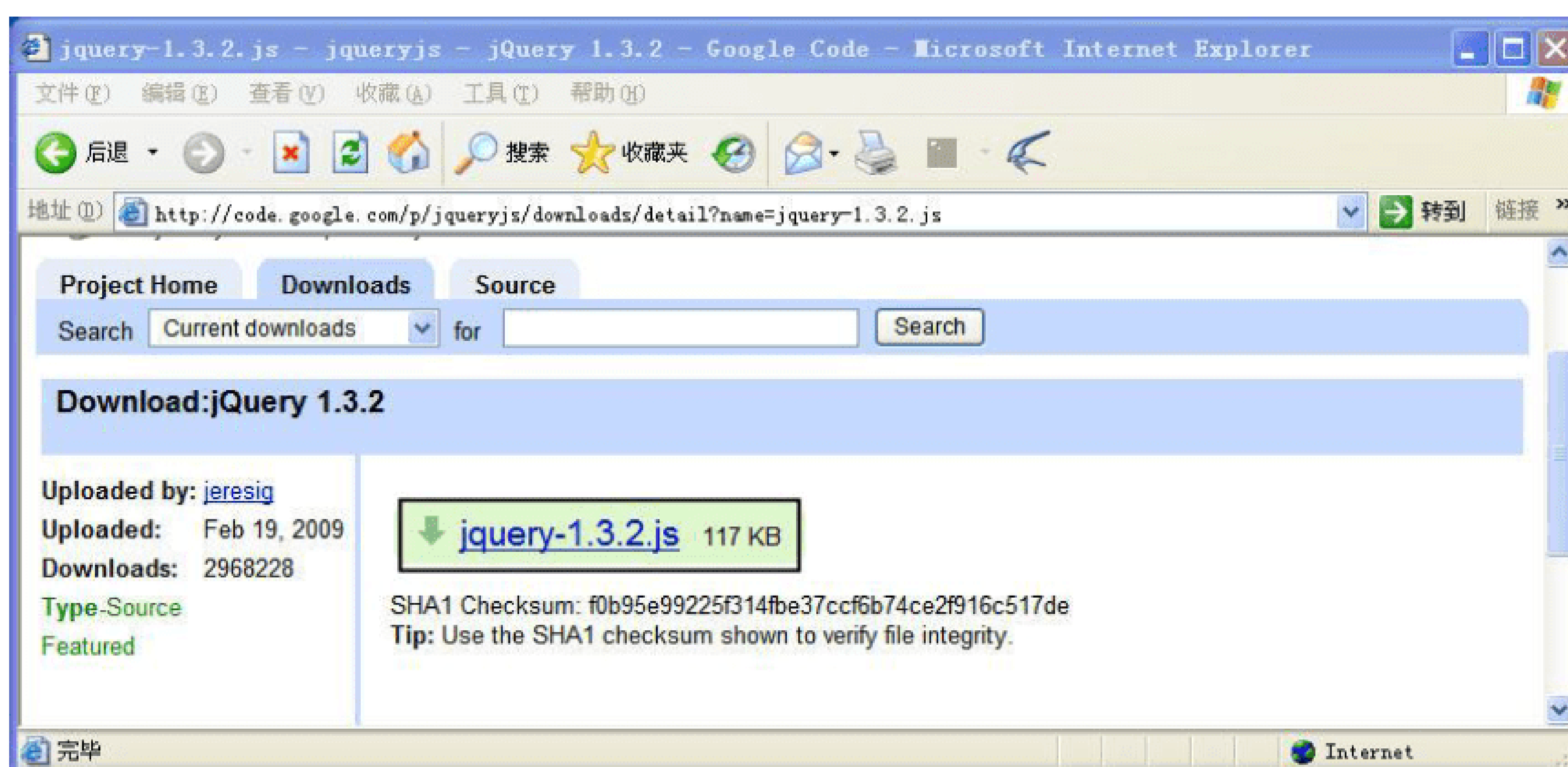


图 14.3　实现 JQuery 组件的下载

至此，就完成对 JQuery 组件的下载。由于该组件是一个 Javascript 文件，所以在具体使用时，可以通过直接引入外部 Javascript 文件的方式来调用该组件。

14.1.3　JQuery 框架的简单使用

本节中，为了能够讲清楚 JQuery 框架，将把本书第 2 章名为 AJAX 的项目修改成由 JQuery 框架来实现的名为 Jqueryajax 项目。虽然通过核心对象 XMLHTTPRequest 也可以实现 AJAX 技术的各种功能，但是使用 JQuery 框架却可以更简单地实现相应功能。具体步骤如下。

（1）新建一个名为 Jqueryajax 的 Web Project，在目录 Jqueryajax/WebRoot 下创建一个名为 Javascript 的文件夹，用来存放该项目的 Javascript 类型文件。

（2）复制 JQuery 框架中的 jquery.js 文件到 Javascript 文件夹中，完整项目的目录结构如图 14.4 所示。

（3）在目录 Jqueryajax/WebRoot 下，首先创建一个名为 ajax.html 的页面。代码 14.1 是信息输入页面主要部分。接着在 Jqueryajax/Src 目录下创建一个名为 ajaxservlet 的 Servlet 程序，具体内容如代码 14.2 所示，由于这两段代码与 AJAX 项目的 ajax.html 和 ajaxservlet 内容完全相同，所以不详细讲解。

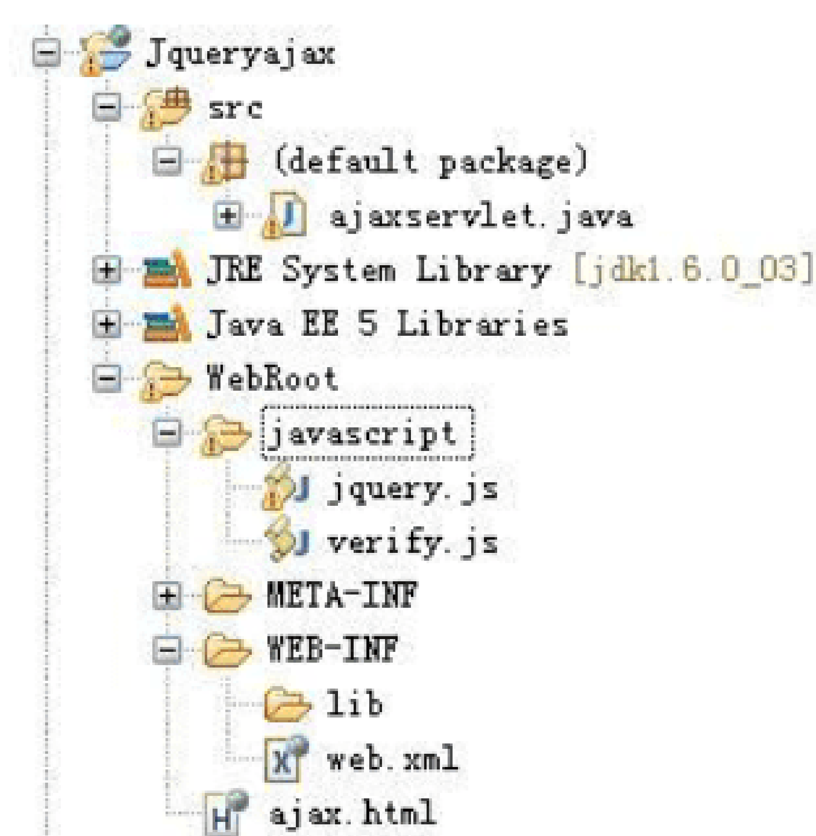


图 14.4　目录结构

代码 14.1　信息输入页面：ajax.html

```
...
    <head>
        <title>用户名校验</title>
        <!--script 语句 -->
        <script type="text/javascript" src="javascript/jquery.js">
        </script>
        <script type="text/javascript" src="javascript/verify.js">
        </script>
    </head>
    <body>
        请输入用户名：
        <!--用户名输入框-->
        <input type="text" id="userName" />
        <!--提交按钮-->
        <input type="button" value="校验" onclick="verify()" />
        <div id="result"></div>
    </body>
...
```

代码 14.2　处理传递参数：ajaxservlet.java

```
...
public class ajaxservlet extends HttpServlet {
    protected void doPost(HttpServletRequest httpServletRequest,
            HttpServletResponse httpServletResponse) throws Servlet
            Exception,
            IOException {
        doGet(httpServletRequest, httpServletResponse);
    }
    //编写 doGet()方法
    protected void doGet(HttpServletRequest httpServletRequest,
            HttpServletResponse httpServletResponse) throws Servlet
            Exception,
            IOException {
        try {
            //设置编码格式

    httpServletResponse.setContentType("text/html;charset=utf-8");
            PrintWriter out = httpServletResponse.getWriter();
            //获取参数
            String old = httpServletRequest.getParameter("name");
            String name = URLDecoder.decode(old, "UTF-8");
```

```
            if (old == null || old.length() == 0) {            //判断参数
                out.println("用户名不能为空");
            } else {
                if (name.equals("cjgong")) {
                    out.println("用户名[" + name + "]已经存在，请使用其他用
                    户名");
                } else {
                    out.println("用户名[" + name + "]尚未存在，可以使用该用户
                    名注册");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

接着在 web.xml 文件中配置 Servlet 程序，具体内容如下。

```
   <!--配置 ajaxservlet 类路径-->
   <servlet>
      <servlet-name>ajaxservlet</servlet-name>
      <servlet-class>ajaxservlet</servlet-class>
</servlet>
   <!--配置 ajaxservlet 映射路径-->
    <servlet-mapping>
      <servlet-name>ajaxservlet</servlet-name>
      <url-pattern>/ajaxservlet</url-pattern>
   </servlet-mapping>
```

（4）创建好客户端和服务器端程序后，接着就需要通过创建一个 JavaScript 文件来实现两者的相互交互，该 Javascript 文件的具体内容如代码 14.3 所示。

代码 14.3　实现交互：verify.js

```
//定义用户名校验的方法
function verify() {
    var jqueryObj = $("#userName");
    var userName = jqueryObj.val();
    $.get("ajaxservlet?name=" + userName, null, callback);
}
//回调函数
function callback(data) {
    var resultObj = $("#result");
    resultObj.html(data);
}
```

【代码解析】

- 由于上述代码需要使用 JQuery 框架的方法，所以必须要把 JQuery 框架中名为 jquery.js 的文件引入该项目。
- 在具体实现客户端与服务器端相互交互时，首先要获取客户端文本框中的内容。查看 JQuery 帮助文档可以发现，通过$()查找节点的方式可以实现该功能。在表达式$()中，参数必须为“#加上 id 属性值”，同时该表达式返回的是 JQuery 对象。接着通过 JQuery 对象的 val()方法获取该节点的值。
- 获取客户端的内容后，就需要把该内容传送给服务器中的 Servlet 程序。通过使用

JQuery 框架中 XMLHTTPrequest 对象的 get()方法把请求封装后发送给服务器，即 $.get()。在该方法中存在 3 个参数，它们分别为处理请求的 URL 地址；发送请求中的 key/value 值对，由于使用 get()方法来发送请求，所以该值一般会直接写在 URL 地址的后面，于是该值一般为 null；回调函数，注意这里只需要函数名称，没有括号。

- 当与服务器交互成功后，就需要调用回掉函数 callback()。在该方法中首先通过参数 data 获取服务器返回的数据，然后再把该数据显示在客户端。在具体显示时，首先获取显示内容节点的 JQuery 对象，然后通过该对象的 html()方法在相应的节点上显示出参数的内容。

（5）单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/Jqueryajax/ajax.html，则会出现如图 14.5 所示的页面。在该页面中如果填写 cjgong 字符串，单击“校验”按钮，该页面显示如图 14.6 所示内容。

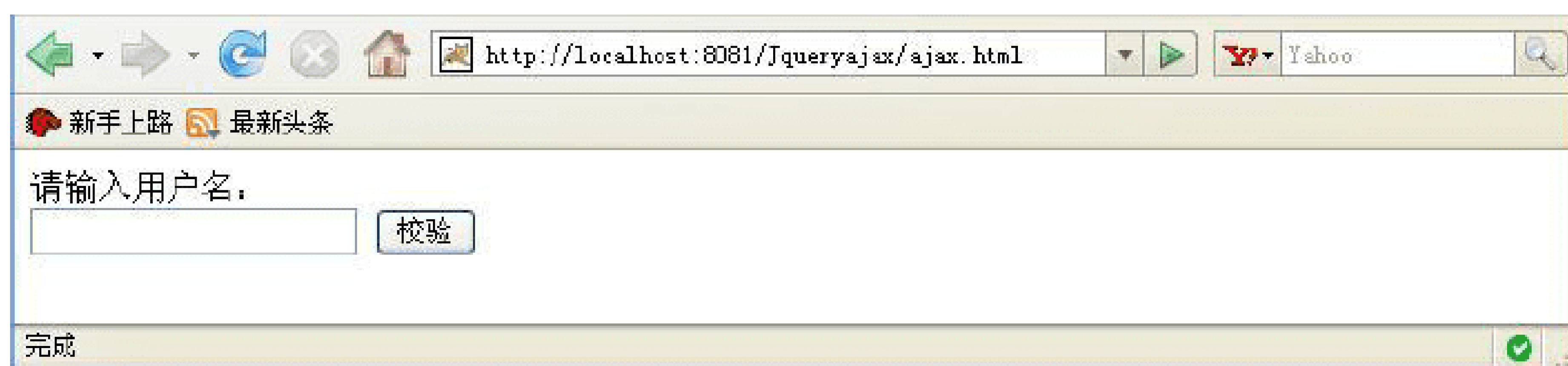


图 14.5　登录首页

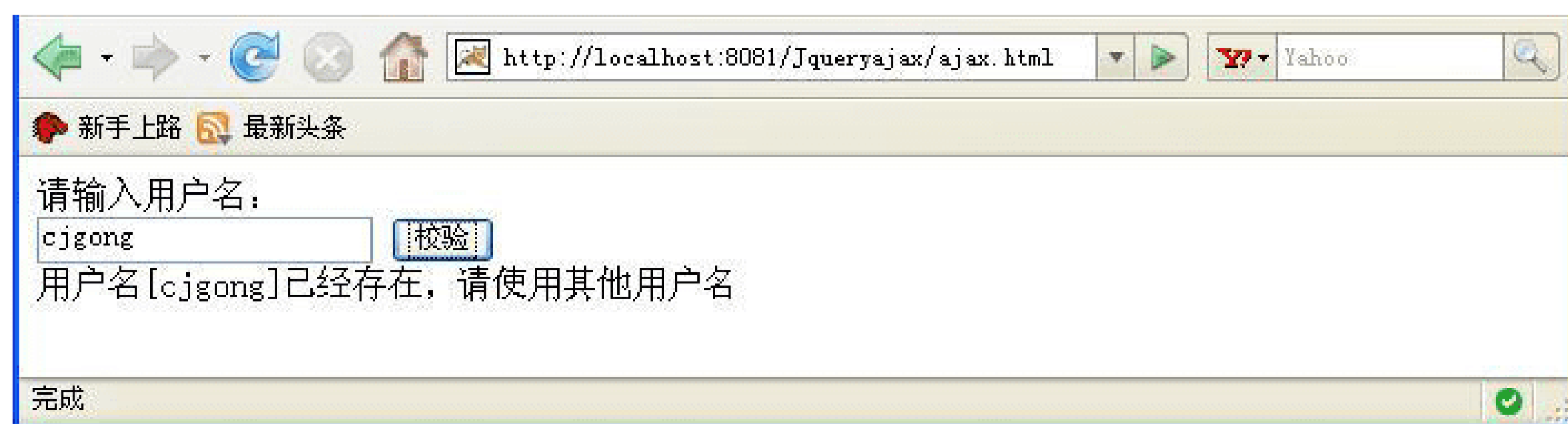


图 14.6　校验结果

14.2　利用 JQuery 框架实现的经典运用

为了深入学习 JQuery 框架，在本节中将实现 3 个关于 JQuery 框架的经典运用，分别为级联菜单、窗口的淡入、淡出和可编辑的边框。

14.2.1　级联菜单

随着时代的发展，网站上需要发布的信息越来越多，所以需要有效地控制网站页面的布局。为了解决多级菜单问题，可以通过 AJAX 技术实现级联菜单来解决该问题。所谓级联菜单，指根据用户的选择，动态地显示出所选菜单的下一级菜单内容。具体效果如下：

当打开页面时只显示出一级菜单（如图 14.7 所示），在该图中单击“我是菜单 1”菜单时，则会出现该菜单的下一级菜单，同理单击“我是菜单 2”菜单时，也会出现该菜单

的下一级菜单（如图 14.8 所示）。如果想收起下一级菜单，可以单击该菜单的上一级菜单。例如当单击图 14.8 中的“我是菜单 1”菜单，则会收起该菜单的下一级菜单（如图 14.9 所示）。



图 14.7　初始菜单

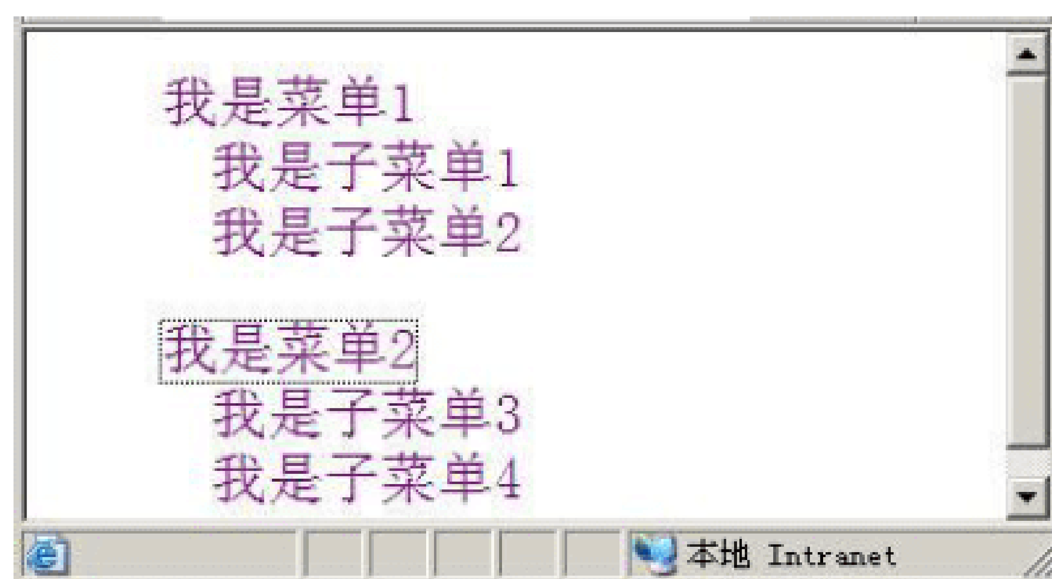


图 14.8　展开菜单效果

为了让每个读者都能够实现级联菜单，下面将详细讲解实现过程中的每一步骤，具体如下。

（1）新建一个名为 ajaxmenu 的 Web Project，在目录 ajaxmenu/WebRoot 下创建一个名为 Javascript 的文件夹，用来存放该项目的 Javascript 类型文件。然后复制 JQuery 框架中的 jquery.js 文件到 Javascript 文件夹中，整个项目的目录结构如图 14.10 所示。



图 14.9　收起菜单效果

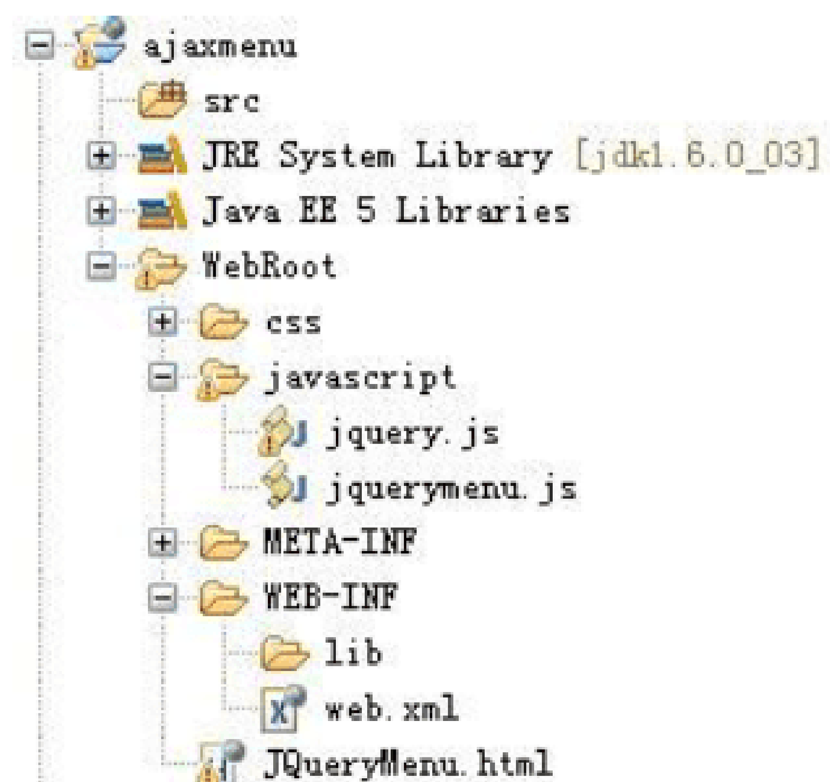


图 14.10　目录结构

（2）在目录 ajaxmenu/WebRoot 下，创建一个名为 JQueryMenu.html 的文件，具体内容如代码 14.4 所示。

代码 14.4　菜单页面：JQueryMenu.html

```
...
<head>
  <title>弹出菜单</title>
  <!--引入 css 文件-->
  <link type="text/css" rel="stylesheet" href="css/menu.css" />
  <!--引入 javascript 文件-->
  <script type="text/javascript" src="javascript/jquery.js"></script>
  <script type="text/javascript" src="javascript/jquerymenu.js"></script>
</head>
<body>
    <ul>
        <a href="#">我是菜单 1</a>
        <li><a href="#">我是子菜单 1</a></li>
        <li><a href="#">我是子菜单 2</a></li>
    </ul>
    <ul>
```

```
        <a href="#">我是菜单 2</a>
        <li><a href="#">我是子菜单 3</a></li>
        <li><a href="#">我是子菜单 4</a></li>
    </ul>
</body>
...
```

注意： 在上述代码中通过 HTML 列表标签<ul>和<li>实现菜单。

（3）在目录 ajaxmenu/WebRoot/Javascript 下，创建一个名为 jquerymenu.js 的 Javascript 文件，具体内容如代码 14.5 所示。

代码 14.5　实现菜单的展开和隐藏：jquerymenu.js

```
$(document).ready( function() {                  //注册页面装载时执行的方法
        var as = $("ul > a");                    //这里需要首先找到所有的主菜单
        as.click( function() {
            //找到当前 ul 中的 li，然后让 li 显示出来
            var aNode = $(this);                 //获取当前被点击的 a 节点
            var lis = aNode.nextAll("li");  //找到当前 a 节点的所有 li 兄弟字节点
            //让子节点显示或隐藏
            lis.toggle("show");
            });
    });
```

【代码解析】

在上述代码中通过 function()方法来实现当单击主菜单的按钮时，对应的子菜单可以显示，再次单击菜单则隐藏功能。该功能主要通过 3 个步骤来实现，分别如下：

首先需要找到所有的主菜单，然后给所有的主菜单注册点击事件，最后通过找到当前 ul 元素中的 li 子元素实现菜单的显示和隐藏。

14.2.2　窗口的淡入、淡出

在 JavaScript 语言中存在一个函数 alert()，该函数可以实现弹出一个窗口。但是弹出的窗口比较古板、不够新颖，本节将实现弹出一个用户创建的窗口，并且在弹出和关闭时实现淡入、淡出。

单击如图 14.11 所示网页中的“显示浮动窗口”链接后，就会在该页面中出现一个窗口，如图 14.12 所示。单击该窗口的关闭按钮后，就会实现窗口的关闭（如图 14.13 所示）。窗口的弹出和关闭都不是一下子实现的，还是实现了淡入、淡出的特殊效果。

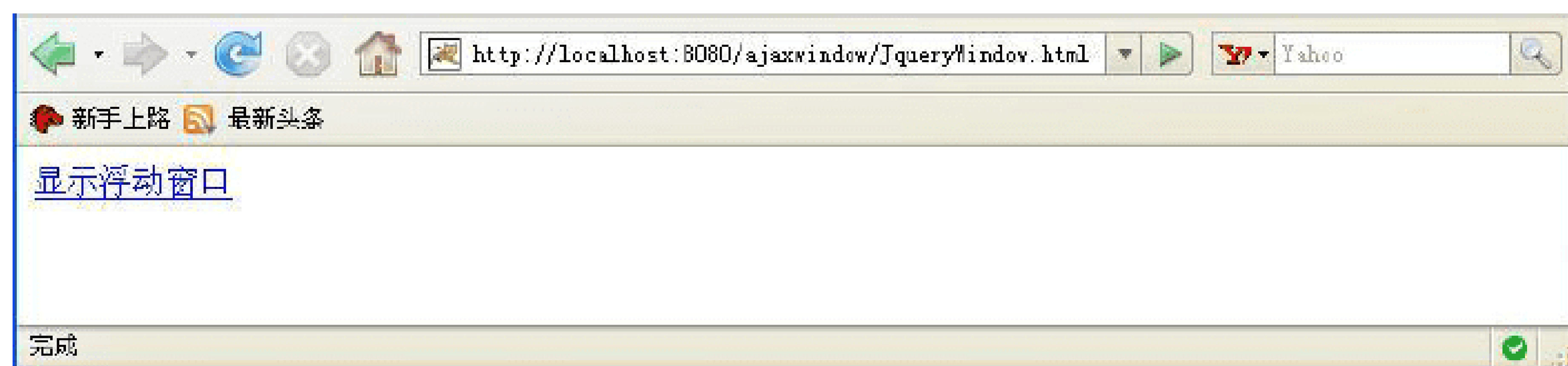


图 14.11　首页

为了让每个读者都能够实现窗口的淡入、淡出，下面将详细讲解实现过程中的每一步骤，具体如下。

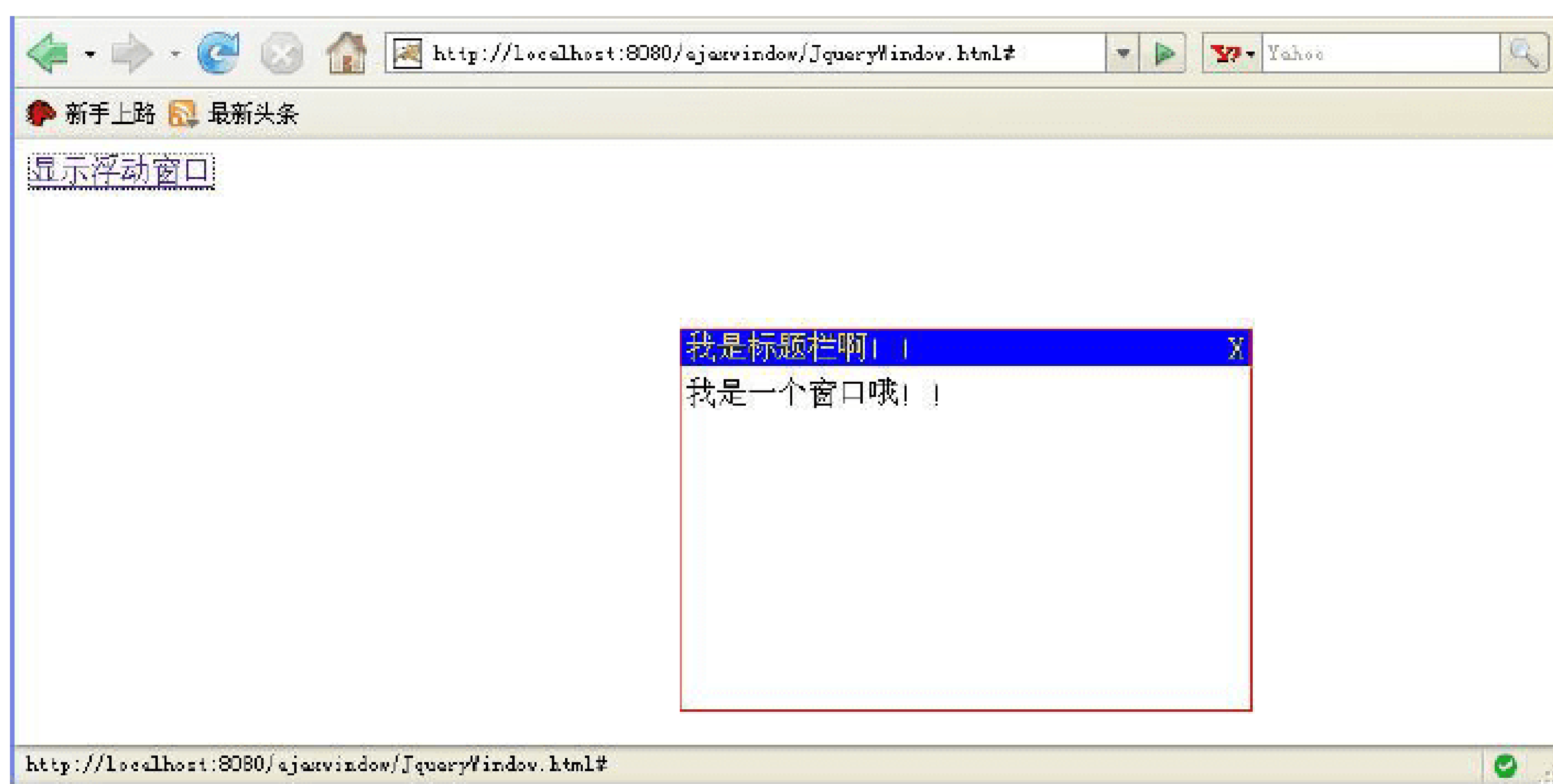


图 14.12　显示窗口

（1）新建一个名为 ajaxwindow 的 Web Project，在目录 ajaxwindow/WebRoot 下创建一个名为 Javascript 的文件夹用来存放该项目的 JavaScript 类型文件。然后复制 JQuery 框架中的 jquery.js 文件到 Javascript 文件夹里，整个项目的目录结构如图 14.14 所示。

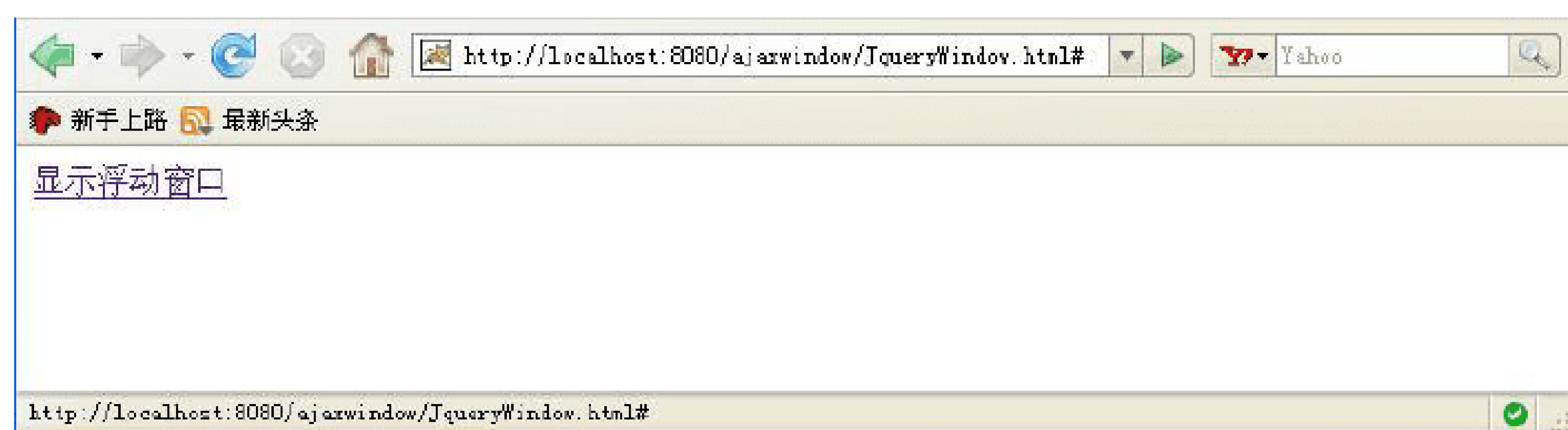


图 14.13　关闭窗口

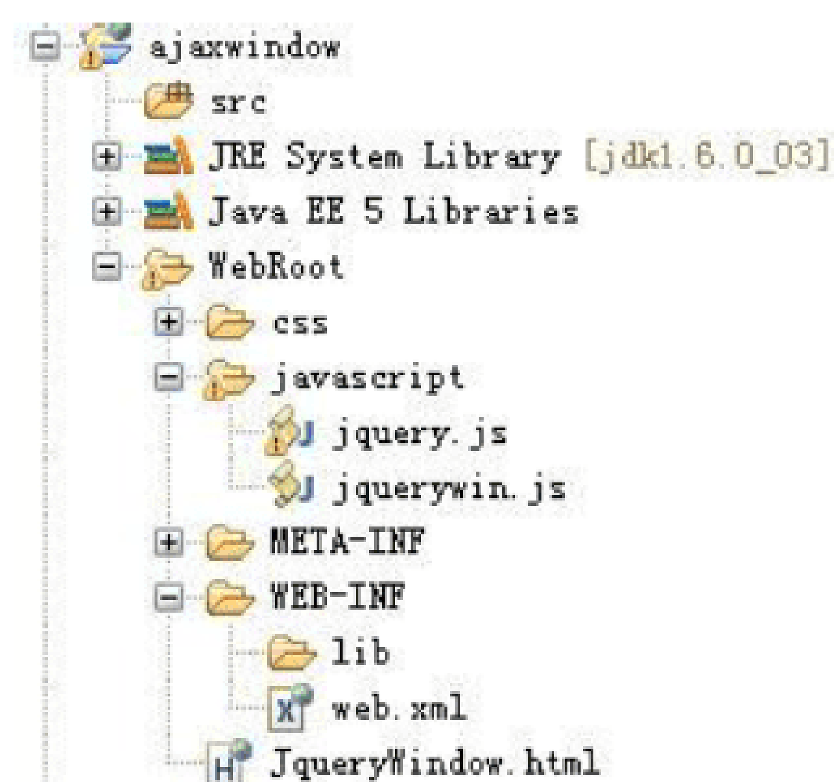


图 14.14　目录结构

（2）在目录 ajaxwindow/WebRoot 下，创建一个名为 JqueryWindow.html 的文件，具体内容如代码 14.6 所示。

代码 14.6　显示窗口页面：JqueryWindow.html

```
...
<head>
  <title>浮动窗口</title>
  <!--链接外部的 js 文件-->
  <script type="text/javascript" src="javascript/jquery.js"></script>
  <script type="text/javascript" src="javascript/jquerywin.js"></script>
  <!--链接外部的 css 文件-->
  <link type="text/css" rel="stylesheet" href="css/win.css" />
</head>
<body>
    <a onclick="showwin()" href="#">显示浮动窗口</a>
    <div id="win">
        <div id="title">我是标题栏啊！！<span id="close" onclick="hide()">
       X</span></div>
        <div id="content">我是一个窗口哦！！</div>
    </div>
</body>
...
```

注意：在上述代码中通过 HTML 列表标签<div>模拟的“浮动窗口”中，为了便于编写，用字符 X 表示窗口的关闭按钮。

（3）在目录 ajaxwindow/WebRoot/Javascript 下，创建一个名为 jquerywin.js 的 Javascript 文件，具体内容如代码 14.7 所示。

代码 14.7　实现窗口的淡入淡出：jquerywin.js

```
function showwin() {                          //显示浮动窗口的方法
    var winNode = $("#win");                  //找到窗口对应的 div 节点
    winNode.fadeIn("slow");                   //调用 fadeOut()方法使窗口显示出来
}
function hide() {                             //隐藏窗口的方法
    var winNode = $("#win");                  //1.找到窗口对应的节点
    winNode.fadeOut("slow");                  //调用 fadeOut()方法使窗口隐蔽起来
}
```

【代码解析】

在上述代码中创建了两个方法：showwin()和 hide()，前者用来实现显示浮动窗口，后者用来实现隐蔽浮动窗口。

14.2.3　可编辑边框

在以前的项目中，在具体实现数据库增、删、改、查（CRUD）时，经常会用表格显示出数据库的每条记录，并且在每条记录的后面添加一些修改、删除等按钮来实现相应功能。在该种情况下，如果想修改数据库数据，必须通过单击“修改”按钮在出现的页面中进行修改。为了简化该过程，可以通过能编辑的边框来显示数据库数据。

打开显示数据页面（如图 14.15 所示）后，如果想修改某个边框的内容，只要双击该边框，在该变长的边框里修改数据（如图 14.16 所示），然后按下 Enter 键，页面显示的就是修改后的数据（如图 14.17 所示）。

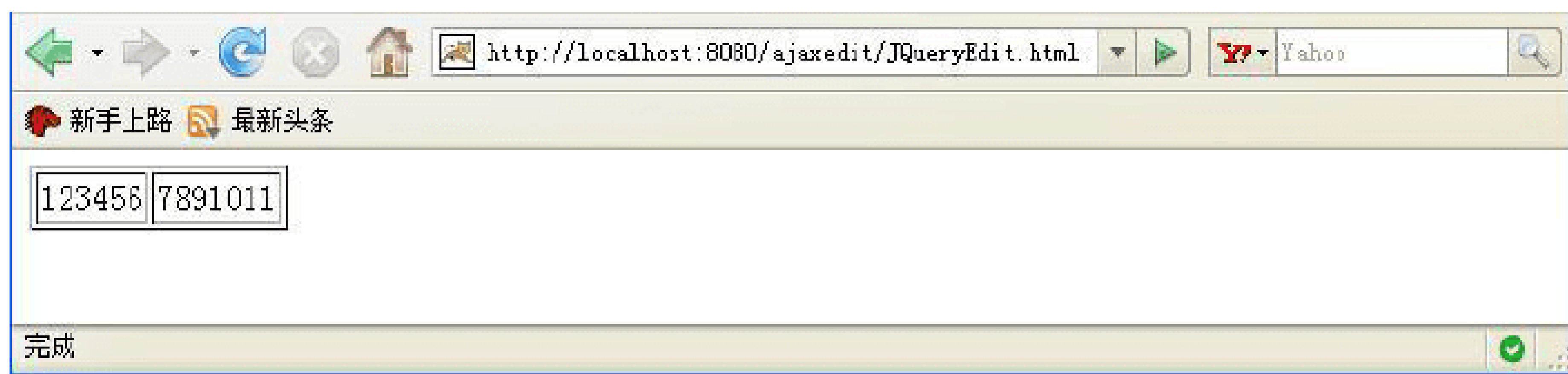


图 14.15　首页

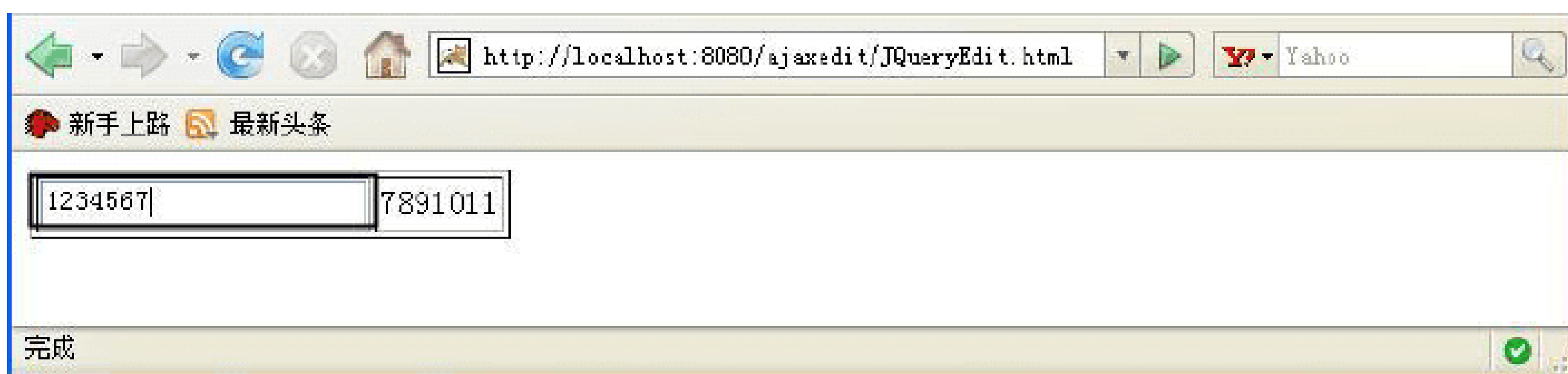


图 14.16　显示窗口

为了让每个读者都能够实现可编辑边框，下面将详细讲解实现过程中的每一步骤，具体如下。

（1）新建一个名为 ajaxedit 的 Web Project，在目录 ajaxedit/WebRoot 下创建一个名为

javascript 的文件夹用来存放该项目的 JavaScript 类型文件。然后复制 JQuery 框架中的 jquery.js 文件到 javascript 文件夹里，整个项目的目录结构如图 14.18 所示。

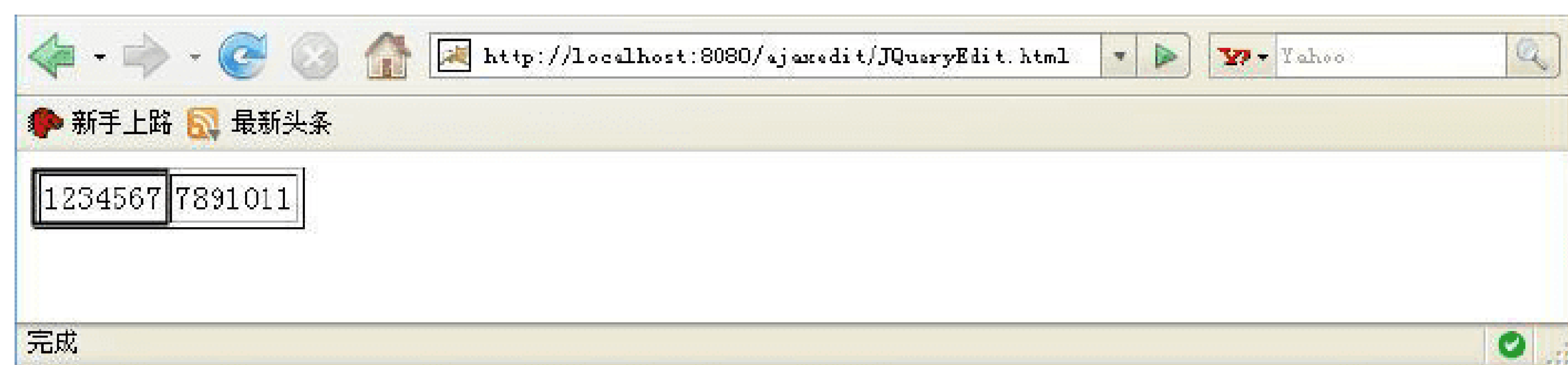


图 14.17　关闭窗口

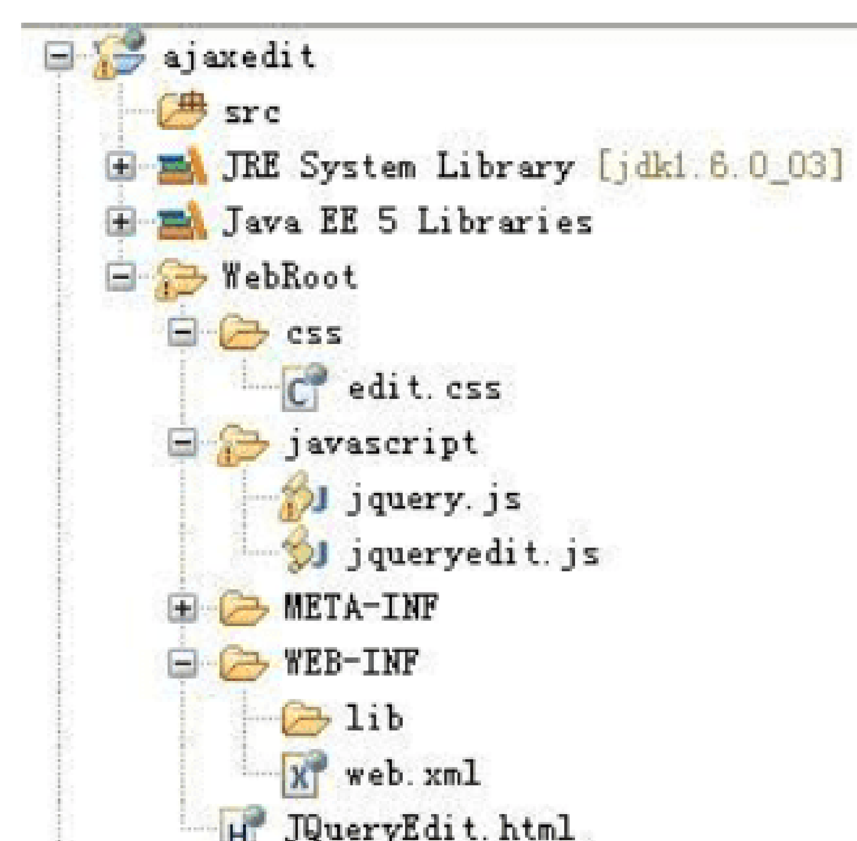


图 14.18　目录结构

（2）在目录 ajaxedit/WebRoot 下，创建一个名为 JQueryEdit.html 的文件，具体内容如代码 14.8 所示。

代码 14.8　显示数据页面：JQueryEdit.html

```
...
    <head>
        <title>可以编辑的边框</title>
        <!--引入 javascript 文件-->
        <script type="text/javascript" src="javascript/jquery.js">
        </script>
        <script  type="text/javascript"  src="javascript/jqueryedit.js">
        </script>
    </head>
    <body>
        <!--一个最简单的表格，一行两列，我们需要让表格中的数据点击时可以修改-->
        <table border="1px">
            <tbody>
                <tr>
                    <td>
                        123456
                    </td><td>
                        7891011
                    </td></tr>
            </tbody>
        </table>
    </body>
...
```

【代码解析】

在上述代码中为了便于讲解，在页面中不仅通过 HTML 表格标签<table>显示数据，而且显示的数据为人为编写的。在真实项目中，数据应该从数据库中获取。

（3）在目录 ajaxwindow/WebRoot/Javascript 下，创建一个名为 jqueryedit.js 的 JavaScript 文件，具体内容如代码 14.9 所示。

代码 14.9　实现可编辑边框：jqueryedit.js

```
$(document).ready( function() {
                                //在页面装载时，让所有的 td 都拥有一个点击事件
```

```
        var tds = $("td");            //找到所有的 td 节点
        tds.click(tdclick);           //给所有的 td 节点增加点击事件
    });
function tdclick() {                  //实现 td 被点击的事件
    var td = $(this);                 //保存当前的 td 节点
    var text = td.text();             //取出当前 td 中的文本内容保存起来
    td.html("");                      //清空 td 中的内容
    var input = $("<input>");         //建立一个文本框，也就是 input 的元素节点
    input.attr("value", text);        //设置文本框的值是保存起来的文本内容
    //让文本框可以响应键盘按下并弹起的事件，主要用于处理回车确认
    input.keyup( function(event) {
        //获取当前用户按下的键值
            //解决不同浏览器获取事件对象的差异
            var myEvent = event || window.event;
            var kcode = myEvent.keyCode;
            //判断是否是回车按下
            if (kcode == 13) {
                var inputnode = $(this);
                var intputext = inputnode.val();    //保存当前文本框的内容
                var tdNode = inputnode.parent();    //清空 td 中的内容
                tdNode.html(intputext);     //将保存的文本框的内容填充到 td 中
                tdNode.click(tdclick);      //让 td 重新拥有点击事件
            }
        });
    td.append(input);                       //将文本框加入到 td 中
    //让文本框中的文字被高亮选中
    //需要将 jquery 的对象转换成 dom 对象
    var inputdom = input.get(0);
    inputdom.select();
    td.unbind("click");                     //需要清除 td 上的点击事件
}
```

【代码解析】

在上述代码中有两个函数：read 和 tdclick，前者为页面加载时调用的函数，实现让所有的 td 都拥有一个点击事件；后一个函数首先实现把表格里的内容显示在 input 表单文本框元素中，然后实现 input 文本内容的修改，最后响应 input 文本框的 Enter 键。

14.3　实现仿 Google Suggest 功能

Google 公司推出了实现简单搜索功能的 Google Suggest 服务，所谓 Google Suggest 服务是指当在搜索框中输入要搜索的词时，Google 则会给出一些相关搜索词的提示，然后通过光标移动来定位。该特效是由 AJAX 来实现。

14.3.1　效果演示

首先打开首页，该页面同 Baidu 和 Google 的页面极为相似，如图 14.19 所示。在搜索页面的文本框中输入“西”字符时，在文本框下面会出现下拉提示框，用来列举出提示的

选项，默认为下拉框的第一选项（如图 14.20 所示）。

如果在下拉提示框中包含选项时，可以通过光标选择相应的选项（如图 14.21 所示），然后按下 Enter 键就可以把选中的内容显示在文本框中，同时下拉提示框消失，如图 14.22 所示。



图 14.19　搜索页面



图 14.20　出现下拉提示框

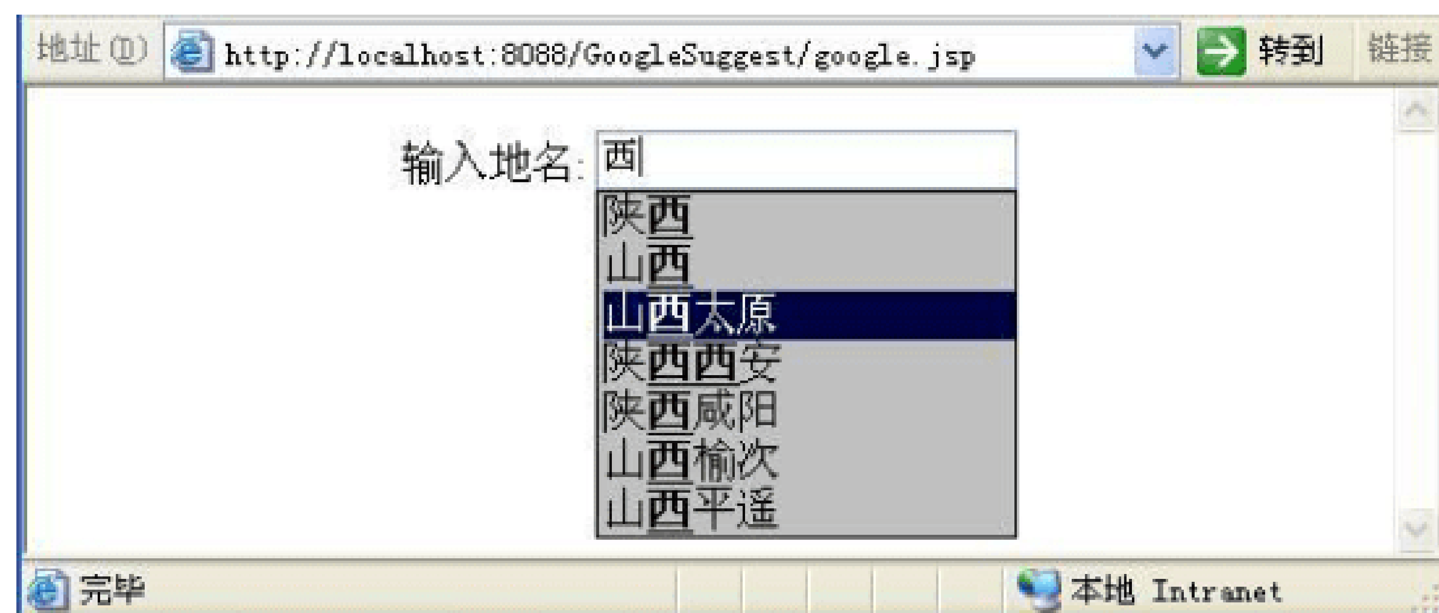


图 14.21　利用光标选择相应选项



图 14.22　Enter 键效果

14.3.2　数据库设计

仿 Google Suggest 功能的项目中需要建立一个数据库和在该数据库中建立一张表，分别为存放表格的数据库 GoogleSuggest 和存放单位信息的表 QINDEX。

1．创建数据库GoogleSuggest

如果想创建出数据库 GoogleSuggest，可以在 Server SQL 2000 的命令行窗口中输入如下命令：

```
CREATE DATABASE 'GoogleSuggest'
```

2．创建表格QINDEX

如果想创建出表 QINDEX，可以在 Server SQL 2000 的命令行窗口中输入如下命令：

```
CREATE TABLE 'QINDEX' (
  'id' int(4) NOT NULL,
  'Content varchar(50)
)
```

该表格的模拟数据如图 14.23 所示。

该表格的具体信息如表 14.1 所示。

ID	Content
1	陕西
2	山西
3	北京
4	北京黑石头
5	山西太原
6	陕西西安
7	陕西咸阳
8	山西榆次
9	山西平遥

图 14.23　orderinfo 表格的模拟数据

表 14.1　表QINDEX信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	int(4)	编号
Content	varchar(50)	检索内容

接着为表格 QINDEX 创建一个实体类文件，实现与数据库字段的对应，具体内容如代码 14.10 所示。

代码 14.10　创建实体对象：Gindex.java

```
...
public class GIndex {
    //创建字段
    private int GID;
    private String GContent;
    public GIndex() {                                        //构造函数
    }
    public GIndex(int id, String content) {
        this.GID = id;
        this.GContent = content;
    }
    //配置属性 GID 和 Gcontent
    ...
}
```

为了便于与数据库连接，专门创建了一个名为 DataQuery 类来返回一个数据库连接对象，具体内容如代码 14.11 所示。

代码 14.11　实现数据库连接：DataQuery.java

```
...
public class DataQuery {
    public DataQuery() {                                        //构造函数
    }
    public Connection getConnection() throws Exception {    //返回连接对象
        //设置驱动类
        String CLASSFORNAME = "com.microsoft.jdbc.sqlserver.SQLServer
        Driver";
        //设置连接参数
        String SERVANDDB = "jdbc:microsoft:sqlserver://localhost:1433;
        DatabaseName=GoogleSuggest";
        String USER = "sa";                                 //设置用户名
        String PWD = "root";                                //设置密码
        Connection con;
```

```
        try {
            Class.forName(CLASSFORNAME);                //加载驱动器
            //获取连接对象
            con = DriverManager.getConnection(SERVANDDB, USER, PWD);
            return con;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

至此，就完成对数据库和表格的设计。

14.3.3　实现查询数据库

在仿 Google Suggest 功能的项目中，需要根据参数值访问数据库并返回查询到的结果，IndexManager 类实现根据参数值访问数据并把查询结果存储到 ArrayList 类型对象中，该类的具体内容如代码 14.12 所示。

代码 14.12　获取数据库数据：IndexManager.java

```
public class IndexManager {
    public ArrayList getManagerList(String param) throws Exception {
        //创建 SQL 语句
        String query = "select top 10 * from QINDEX where Content like '%"
                + param + "%'";
        DataQuery dq = new DataQuery();                 //创建 DataQuery 对象
        Connection con = dq.getConnection();            //获取 Connection 对象
        ArrayList al = new ArrayList();                 //创建存储数据对象
        try {
            Statement stmt = con.createStatement();         //获取到陈述对象
            ResultSet rs = stmt.executeQuery(query);        //得到执行结果
            //遍历执行结果 rs
            while (rs.next()) {
                int id = rs.getInt("ID");                   //获取 ID 的值
                String content = rs.getString("Content");
                                                            //获取 Content 的值
                GIndex gindex = new GIndex(id, content);
                                                            //创建 Gindex 对象
                al.add(gindex);                         //添加到 ArrayList 类型对象中
            }
            rs.close();
            stmt.close();
            con.close();
            return al;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

14.3.4　实现对请求处理

在仿 Google Suggest 功能的项目中，当客户端发出请求后会被名为 GetIndexServlet 的 Servlet 程序来处理，该 Servlet 程序的具体内容如代码 14.13 所示。

代码 14.13　对请求处理：GetIndexServlet.java

```
public class GetIndexServlet extends HttpServlet {
    //编写 doGet()方法
    public  void  doGet(HttpServletRequest  request,  HttpServletResponse
    response)
            throws IOException, ServletException {
        response.setContentType("text/xml;charset=UTF-8");
                                                    //设置编码格式
        response.setHeader("Cache-Control", "no-cache");
                                                    //设置是否需要缓冲
        //获取参数的值并对其进行编码格式的设置
        String param = (String) request.getParameter("param");
        param = new String(param.getBytes("ISO8859 1"));
        ArrayList alist = new ArrayList();          //创建 ArrayList 对象
        try {
            //调用 IndexManager 类的 getManagerList()方法获取查询结果
            alist = (new IndexManager()).getManagerList(param);
        } catch (Exception e) {
        }
        //以 XML 文件格式输出查询结果
        String xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";
        xml += "<message>";
        Iterator iter = alist.iterator();
        String content;
        while (iter.hasNext()) {
            GIndex gindex = (GIndex) iter.next();
            content = gindex.getContent();
            xml += "<info>" + content + "</info>";
        }
        xml += "</message>";
        response.getWriter().write(xml);
    }
    //编写 doPost()方法
    public  void  doPost(HttpServletRequest  request,  HttpServletResponse
    response)
            throws IOException, ServletException {
        this.doGet(request, response);
    }
}
```

接着在 web.xml 文件中实现对该 Servlet 类的配置。

```
    <!--配置 makeIndex 类路径-->
    <servlet>
        <servlet-name>makeIndex</servlet-name>
        <servlet-class>com.cjg.servlet.GetIndexServlet</servlet-class>
    </servlet>
    <!--配置 makeIndex 映射路径 -->
    <servlet-mapping>
        <servlet-name>makeIndex</servlet-name>
```

```
        <url-pattern>/make</url-pattern>
    </servlet-mapping>
```

注意： 在上述代码中，最终返回一个 XML 格式字符串，在该字符串中包含了从数据库中查询到的匹配数据。

14.3.5　Google Suggest 功能的客户端页面

在仿 Google Suggest 功能的项目中，关于客户端的文件一共有 3 个，分别为 index.jsp、style.css 和 build.js。其中 index.jsp 页面用来实现用户输入查询条件。

index.jsp 页面用来发出请求，该页面的具体内容如代码 14.14 所示。

代码 14.14　首页：index.jsp

```
...
    <head>
        <title>Google Suggest</title>
        <!--引入外部 js 文件-->
        <script src="build.js" language="javascript"></script>
    </head>
    <body onResize="ReDraw()">
        <div align="center">
            <!--表单-->
            <form name="Form1" AUTOCOMPLETE="off" ID="Form1">
                AutoComplete Text Box:
                <input type="text" name="txtUserInput" /> <!--文本输入框-->
                <!--隐藏域-->
                <input type="hidden" name="txtUserValue" ID="hidden1" />
                <!--辅助文本输入框-->
                <input type="text" name="txtIgnore" style="display:
                none" />
            </form>
        </div>
    </body>
...
```

【代码解析】

在上述代码中为什么会有两个输入框和一个隐藏域呢？名为 txtUserInput 的文本框用来接受用户的输入，隐藏域用来显示数据库返回的查询字符串，而名为 txtIgnore 的文本框用来防止当用户按下 Enter 键时向服务器端提交表单。

14.4　Google Suggest 功能的相关 JavaScript 代码

在仿 Google Suggest 功能的项目中，关于客户端的文件一共有 3 个，分别为 index.jsp、style.css 和 build.js。其中 build.js 文件存放关于该项目的所有 JavaScript 语句。该文件内容的基本流程如图 14.24 所示。

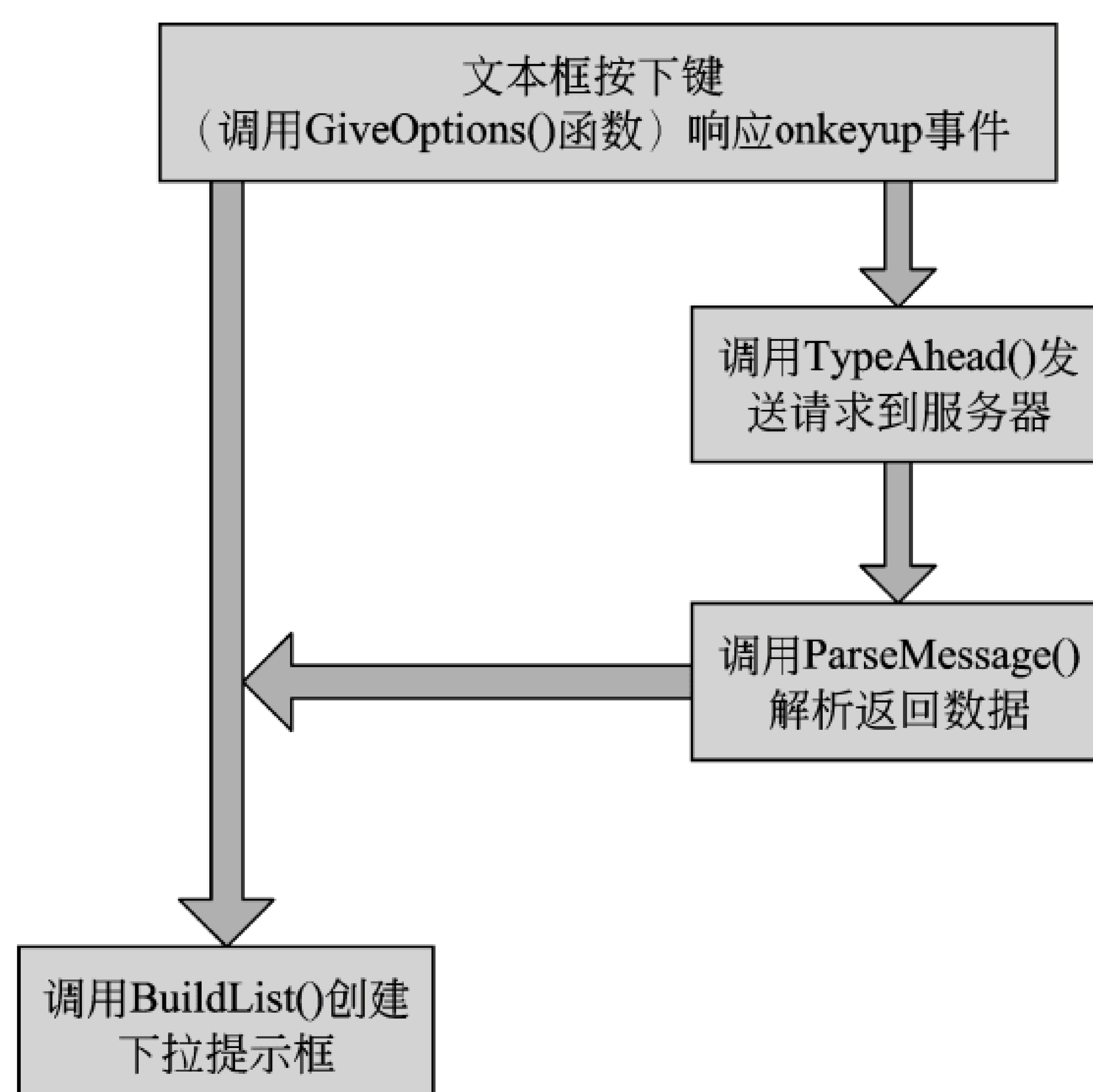


图 14.24　客户端流程

14.4.1　判断按键的 JavaScript 代码——GiveOptions()方法

build.js 页面存放关于 GoogleSuggest 项目的所有用户编写的 JavaScript 代码，在该页面中存在一个名为 GiveOptions()的方法，具体内容如代码 14.15 所示。

代码 14.15　判断按钮方法：build.js

```
var arrOptions = new Array();           //定义一个保存服务端返回的数据数组
var strLastValue = "";                  //定义保存每一次向服务器发送请求的参数
var theTextBox;                         //定义表示文本输入变量
var currentValueSelected = -1;          //定义下拉提示框中默认的选择项
window.onload = function() {            //默认加载方法
    var elemSpan = document.createElement("span");
    elemSpan.id = "spanOutput";
    elemSpan.className = "spanTextDropdown";
    document.body.appendChild(elemSpan);
    document.Form1.txtUserInput.onkeyup = GiveOptions;
}
function GiveOptions() {                //按下按键调用方法
    var intKey = -1;
    if (window.event) {
        intKey = event.keyCode;
        theTextBox = event.srcElement;
    }
    if (theTextBox.value.length == 0) {         //文本框内容为空
        HideTheBox();
        strLastValue = "";
        return false;
    }
    if (intKey == 13) {                         //当按键为“Enter”
        GrabHighlighted();
        theTextBox.blur();
        return false;
```

```
    } else if (intKey == 38) {                    //当按键为“↑”
        MoveHighlight(-1);
        return false;
    } else if (intKey == 40) {                    //当按键为“↓”
        MoveHighlight(1);
        return false;
    }
    //进行内容比较
    if (theTextBox.value.indexOf(strLastValue) !=0 || arrOptions.length==0
            || (strLastValue.length == 0 &&
    theTextBox.value.length > 0)
            || (theTextBox.value.length <=
            strLastValue.length)) {
        strLastValue = theTextBox.value;
        TypeAhead(theTextBox.value);
    } else {
        BuildList(theTextBox.value);
    }
}
```

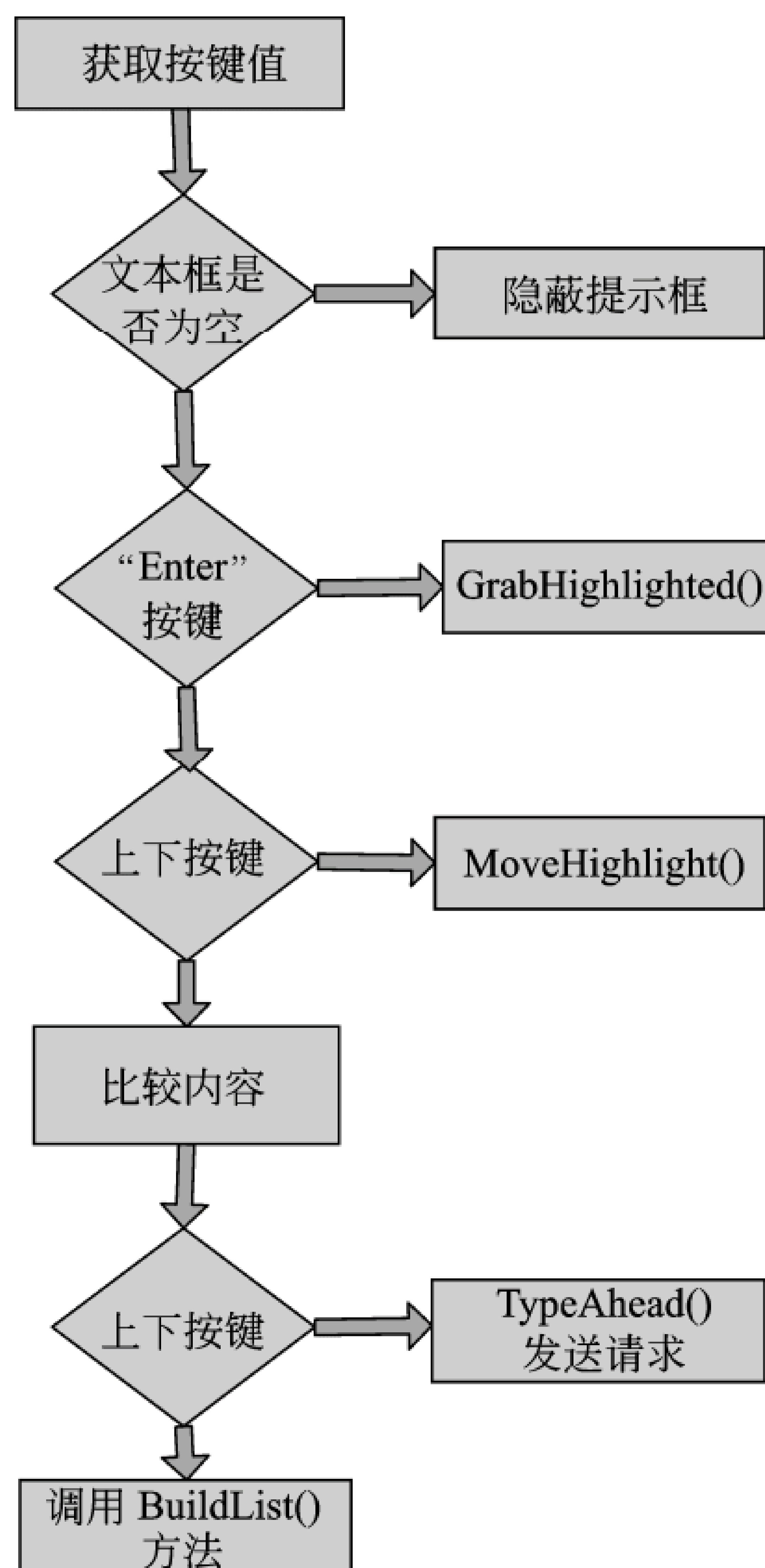


图 14.25　GiveOptions()流程

【代码解析】

当用户在页面的文本框中按下相应的键时，则会触发 GiveOption()方法。该方法的流程如图 14.25 所示，定义一个表示用户输入按键的 Unicode 值的变量 intKey，通过对该值的判断来决定继续调用哪个函数。

在上述代码中，除了 GiveOption()方法外，还存在一个名为 window.onload 的方法，当页面被加载时会自动调用。

14.4.2　解析数据的 JavaScript 代码——parseMessage()方法

build.js 页面存放关于 GoogleSuggest 项目的所有用户编写的 JavaScript 代码，在该页面中存在一个名为 parseMessage()的方法，具体内容如代码 14.16 所示。

代码 14.16　解析数据方法：build.js

```
function TypeAhead(xStrText) {                        //发送请求方法
    var url = "make?param=" + xStrText;               //创建发送地址变量
    if (window.XMLHttpRequest) {                      //判断浏览器
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
    if (req) {
        req.open("GET", url, true);                   //打开连接
        req.onreadystatechange = callback;            //设置回调函数
        req.send(null);                               //实现发送
    }
}
```

```
function callback() {                                        //回调方法
    if (req.readyState == 4) {                               //判断 readyState 码
        if (req.status == 200) {                             //判断 status 码
            parseMessage();
        } else {
            alert("Not able to retrieve description" + req.statusText);
        }
    } else {
    }
}
function parseMessage() {                                    //分析服务器返回数据
    var xmlDoc = req.responseXML.documentElement;  //获取返回的 XML 文件对象
    var node = xmlDoc.getElementsByTagName('info');     //获取标记<info>
    arrOptions = new Array();                               //创建一个数组对象
    for ( var i = 0; i < node.length; i++) {         //存储<info>到数组对象中
        arrOptions[i] = node[i].firstChild.nodeValue;
    }
    BuildList(theTextBox.value);                             //构建提示框
    strLastValue = theTextBox.value;
}
```

【代码解析】

parseMessage()函数主要用来分析服务器返回的数据，该函数的流程如图 14.26 所示。除了该方法外，还定义了两个方法：用于实现发送请求的 TypeAhead()方法和实现回调的 callback()方法。

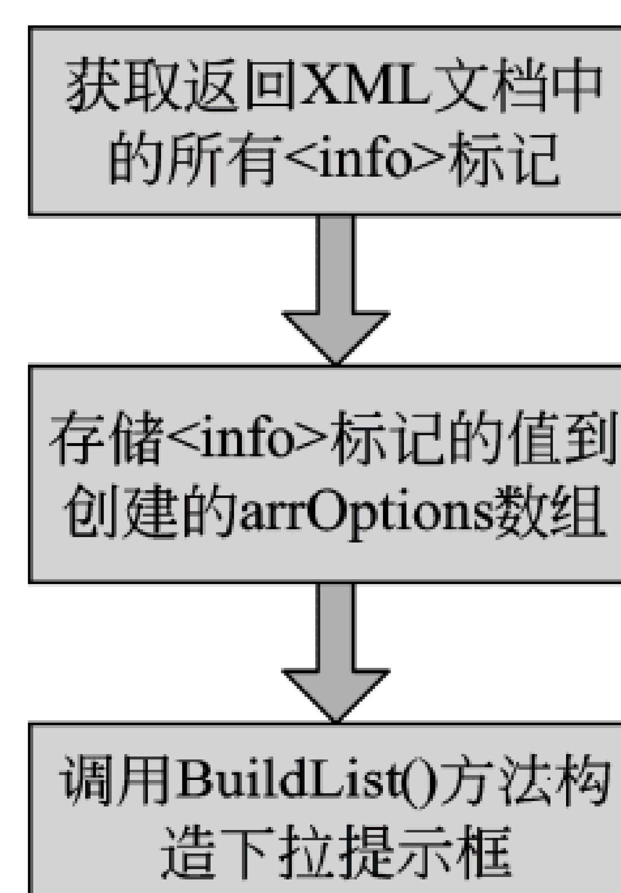


图 14.26　parseMessage()方

14.4.3　创建提示框的 JavaScript 代码——BuildList()方法

build.js 页面存放关于 GoogleSuggest 项目的所有用户编写的 JavaScript 代码，在该页面中存在一个名为 BuildList()的方法，具体内容如代码 14.17 所示。

代码 14.17　创建提示框：build.js

```
function BuildList(theText) {                        //创建下拉提示框方法
    SetElementPosition();                            //调用 SetElementPosition()方法
    var inner = "";                                  //创建一个变量
    var theMatches = MakeMatches(theText); //获取所要匹配的值
    for ( var i = 0; i < theMatches.length; i++) {      //生成 inner 的值
        inner += theMatches[i];
    }
    if (theMatches.length > 0) {                             //生成符合的字符串
        document.getElementById("spanOutput").innerHTML = inner;
        document.getElementById("OptionsList_0").className  =  "spanHigh
        Element";
        currentValueSelected = 0;
    } else {
        HideTheBox();
    }
}
function SetElementPosition() {                              //设置下拉提示框位置方法
```

```
    //创建关于位置的关于 x，y 坐标的变量
    var selectedPosX = 0;
    var selectedPosY = 0;
    //创建关于提示框的长度和宽度的变量
    var theElement = document.Form1.txtUserInput;
    var theTextBoxInt = document.Form1.txtUserInput;
    if (!theElement)                              //判断变量 theElement
        return;
    //为提示框的长度和宽度赋值
    var theElemHeight = theElement.offsetHeight;
    var theElemWidth = theElement.offsetWidth;
    while (theElement != null) {                  //设置提示框的位置
        selectedPosX += theElement.offsetLeft;
        selectedPosY += theElement.offsetTop;
        theElement = theElement.offsetParent;
    }
    xPosElement = document.getElementById("spanOutput");
                                                  //获取元素 spanOutput
    xPosElement.style.left = selectedPosX;        //设置提示框的 left 属性
    xPosElement.style.width = theElemWidth;       //设置提示框的 width 属性
    xPosElement.style.top = selectedPosY + theElemHeight
                                                  //设置提示框的 top 属性
    xPosElement.style.display = "block";
}
```

【代码解析】

BuildList()函数根据 parseMessage()函数返回的结果动态创建下拉提示框，该函数的流程如图 14.27 所示。除了该方法外，还定义了用来设置下拉提示框位置 SetElementPosition()方法。

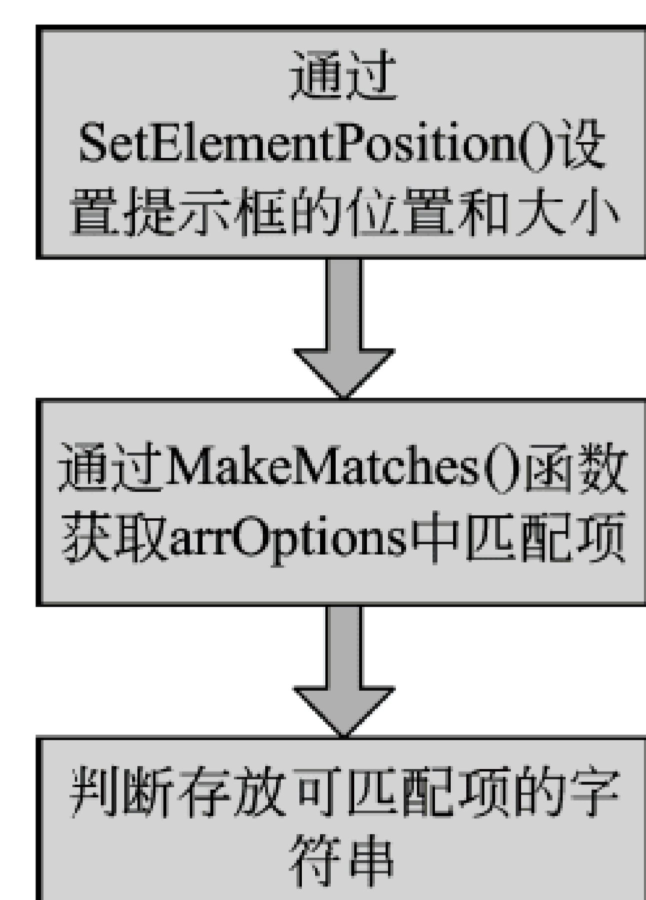


图 14.27　BuildList()方法

14.4.4　查找匹配项的 JavaScript 代码——MakeMatches()方法

build.js 页面存放关于 GoogleSuggest 项目的所有用户编写的 JavaScript 代码，在该页面中存在一个名为 MakeMatches()的方法，具体内容如代码 14.18 所示。

代码 14.18　查找匹配选项：build.js

```
var countForId = 0;                               //定义全局变量
function MakeMatches(xCompareStr) {               //匹配项方法
    //创建变量
    countForId = 0;
    var matchArray = new Array();
    for ( var i = 0; i < arrOptions.length; i++) {
                                                  //遍历对象 arrOptions
        var regExp = new RegExp(xCompareStr, "ig");
                                              //调用 RegExp()方法查看数组
        if ((arrOptions[i].search(regExp)) >= 0) { //判断
            //当有匹配的项，调用 CreateUnderline()方法返回字符串
            matchArray[matchArray.length] = CreateUnderline(arr
            Options[i],
                    xCompareStr, i);
        } else {
```

```
            continue;
        }
    }
    return matchArray;                                    //返回结果
}
```

【代码解析】

MakeMatches()函数用于根据文本框内的值从 parse Message()函数返回结果中，查找出所有可以匹配的项，然后给每个可匹配的项添加一些特殊的标记。该方法的流程如图 14.28 所示。

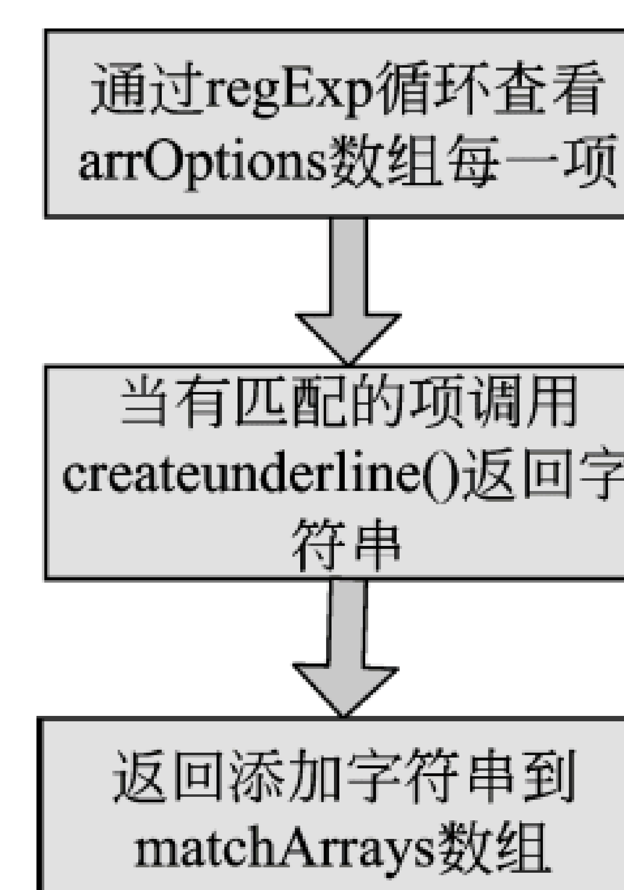


图 14.28　MakeMatches()方法

14.4.5　其他 JavaScript 方法

build.js 页面存放关于 GoogleSuggest 项目的所有用户编写的 JavaScript 代码，在该页面中除了上面几节的方法，各个方法的具体内容如代码 14.19 所示。

代码 14.19　其他方法：build.js

```
var undeStart = "<span class='spanMatchText'>";
var undeEnd = "</span>";
var selectSpanStart = "<span    style='width:100%;display:block;'
class='spanNormalElement' onmouseover='SetHighColor(this)' ";
var selectSpanEnd = "</span>";
function CreateUnderline(xStr, xTextMatch, xVal) {     //定义 HTML 标记方法
    selectSpanMid = "onclick='SetText(" + xVal + ")'" + " id='OptionsList_"
            + countForId + "' theArrayNumber='" + xVal + "'>";
    countForId++;
    var regExp = new RegExp(xTextMatch, "ig");
    var start = xStr.search(regExp);
    var matchedText = xStr.substring(start, start + xTextMatch.length);
    var Replacestr  = xStr.replace(regExp, undeStart + matchedText +
    undeEnd);
    return selectSpanStart + selectSpanMid + Replacestr + selectSpanEnd;
}
function SetHighColor(theTextBox) {                      //设置选项样式方法
    if (theTextBox) {
        currentValueSelected = theTextBox.id.slice(
                theTextBox.id.indexOf(" ") + 1, theTextBox.id.length);
    }
    for ( var i = 0; i < countForId; i++) {
        document.getElementById('OptionsList ' + i).className = 'span
        NormalElement';
    }
    document.getElementById('OptionsList_' + currentValueSelected).class
    Name = 'spanHighElement';
}
function SetText(xVal) {                                 //填入文本框方法
    theTextBox = document.Form1.txtUserInput;
    theTextBox.value = arrOptions[xVal]; //set text value
    document.getElementById("spanOutput").style.display = "none";
    currentValueSelected = -1; //remove the selected index
}
function GrabHighlighted() {                             //Enter 按键方法
```

```
    if (currentValueSelected >= 0) {                          //判断按键的值
        xVal = document.getElementById("OptionsList " + currentValue
        Selected)
                .getAttribute("theArrayNumber");
        SetText(xVal);
        HideTheBox();
    }
}
function HideTheBox() {                                   //隐藏下拉提示框方法
    document.getElementById("spanOutput").style.display = "none";
    currentValueSelected = -1;
}
function MoveHighlight(xDir) {                            //选择键方法
    if (currentValueSelected >= 0) {
        //获取按键的值
        newValue = parseInt(currentValueSelected) + parseInt(xDir);
        if (newValue > -1 && newValue < countForId) {  //判断键的值
            currentValueSelected = newValue;
            SetHighColor(null);
        }
    }
}
function ReDraw() {                                       //重画文本框位置
    BuildList(document.Form1.txtUserInput.value);
}
```

【代码解析】

在上述代码中首先定义了几个 HTML 标记字符串，undStart 和 undeEnd 字符串用来表示每一项中和文本输入框中字符相等的字符串标记；selectSpanStart 和 selectSpanEnd 字符串用来表示下拉框中的每一项的标记。接着定义了方法 CreateUnderline()，该方法需要接收 3 个参数：arrOptions 数组中匹配项的值、文本框中的值和匹配项即 arrOptions 数组中的下标。最后还定义了用来设置下拉提示框中每一项样式的 SetHighColor()方法、用来将选定的项添入文本框中的 SetText()方法、用来实现隐藏下拉提示框的 HideTheBox()方法和用来设置文本框位置变化的 ReDraw()方法。

14.5　小　结

本章主要介绍关于 AJAX 技术 JQuery 框架的经典应用，之所以要讲解 JQuery 框架，因为该框架封装了 AJAX 语法，可以简化 AJAX 的开发。

为了让读者深入理解 JQuery 框架，本章不仅讲解了 JQuery 框架的基础知识外，而且还详细讲解了 3 个典型的应用：级联菜单、窗口的淡入、淡出和可编辑的边框。最后还实现了一个完整仿 Google Suggest 功能的 AJAX 项目。

第 15 章　在线文件上传和下载（Struts 2.x+FileUpload）

一个功能强大的网站系统一般都会包含在线文件上传和下载模块，对于浏览者来说可以通过在线上传模块把本地的文件上传到服务器上保存起来，当需要的时候则可以通过在线下载模块下载下来。

本章将通过 Struts 2.x+Components-FileUpload 框架技术来介绍如何实现在线文件上传和下载，该模块主要包含两种功能：文件上传功能和文件下载功能。

15.1　在线文件上传和下载模块原理

当浏览者想上传文件时，只要打开“选择上传文件”的页面，在该页面选择所要上传的文件就可以实现文件的上传。当浏览者想下载文件时，只要打开“下载文件”的页面，单击相应的链接就可以实现文件的下载。

15.1.1　在线文件上传和下载结构框架分析

对于一个大型网站系统来说，实现一个可用的在线文件上传和下载功能要考虑的情况十分复杂，例如：如何合理规划上传文件的后缀名、如何合理规划上传文件的大小等。本章将会实现一个比较简单可用的在线文件上传和下载功能，读者可以根据自己的需求自行进行完善。本系统结构框架如图 15.1 所示，项目目录如图 15.2 所示。

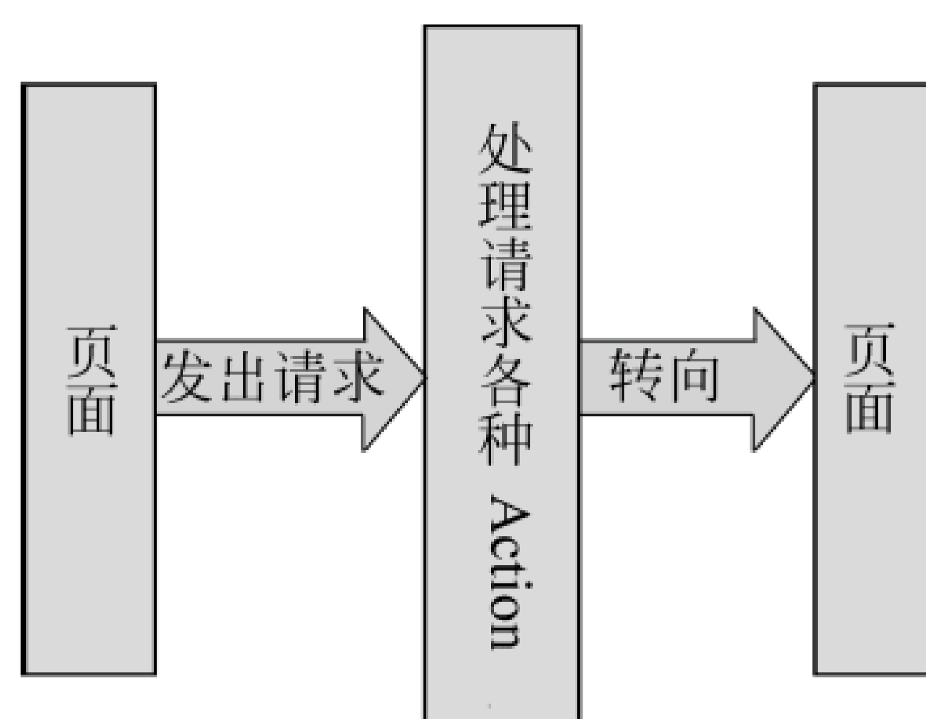


图 15.1　系统流程图

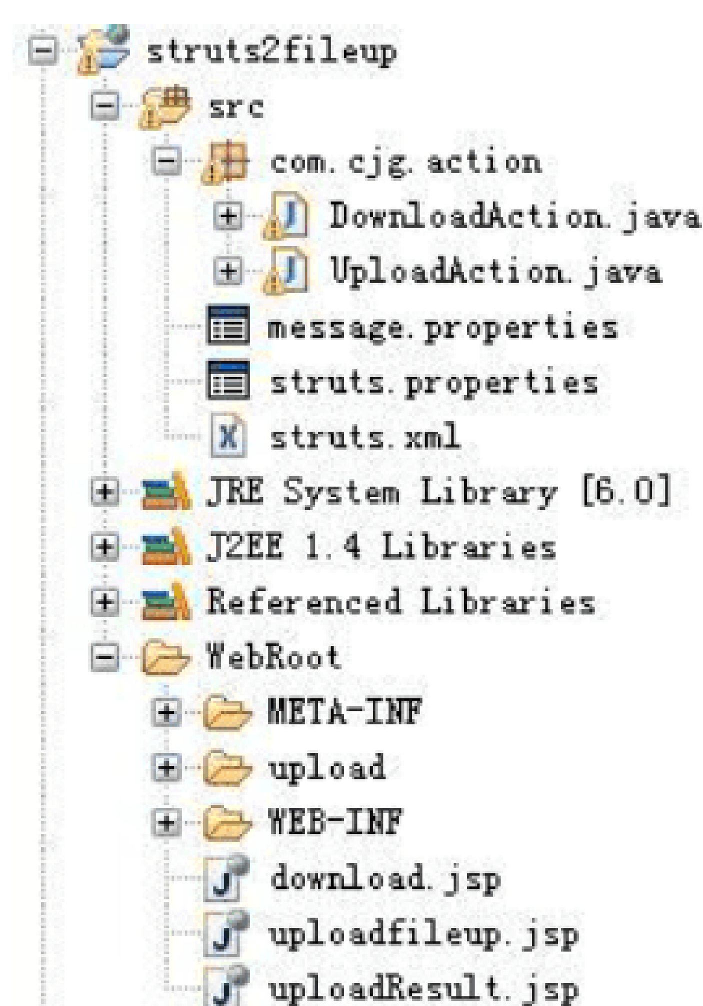


图 15.2　项目目录

15.1.2　在线文件上传和下载功能描述

在本节中，以直观的方式向读者介绍整个在线文件上传和下载模块要实现的功能。这些功能包括单文件的上传、多文件上传和文件下载。

1．单文件上传

浏览者首先通过浏览 uploadfileup.jsp 网页来选择文件，在该页面（如图 15.3 所示）上填写完相应的内容和选择文件后，单击 submit 按钮就可以实现文件上传功能的同时转到显示上传文件信息的页面（如图 15.4 所示）。

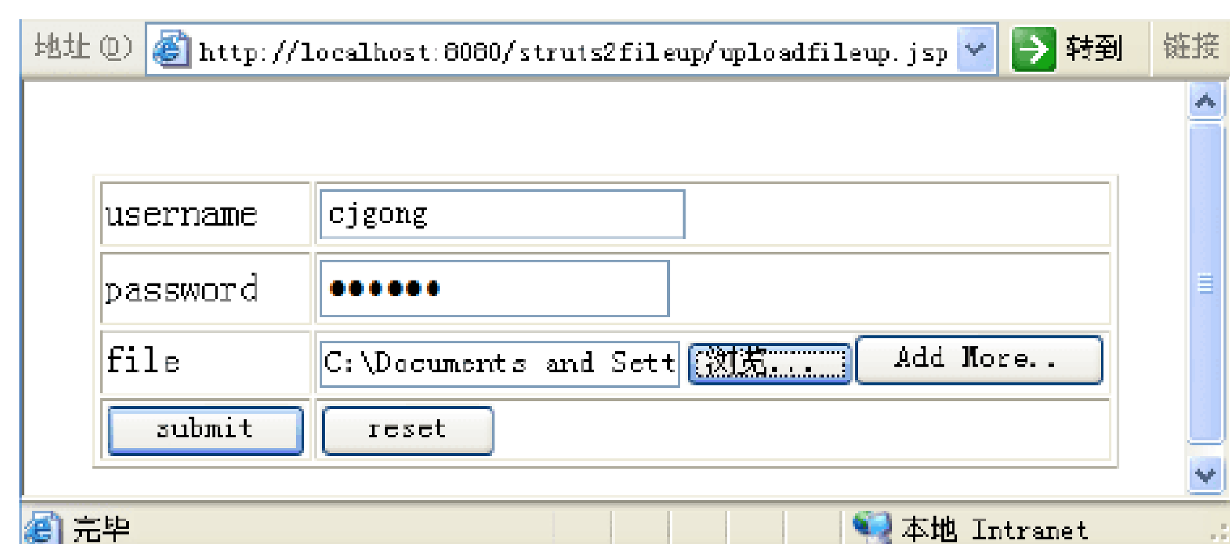


图 15.3　实现文件上传

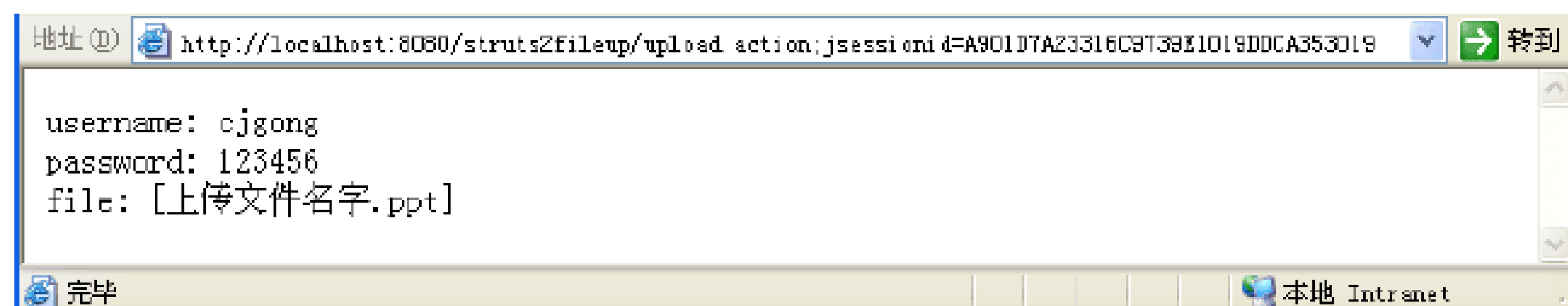


图 15.4　显示上传文件信息

2．多文件上传

当浏览者浏览 uploadfileup.jsp 网页来选择完一个文件后，如果还需要上传其他文件，可以单击 Add More 按钮在出现的 file 文件域中选择相应的文件（如图 15.5 所示）。如果想删除某 file 文件域，只要单击相对应的 Remove 按钮就可以实现，如图 15.6 所示。最后单击 submit 按钮就可以实现多文件上传功能的同时，转到显示上传文件信息的页面（如图 15.7 所示）。

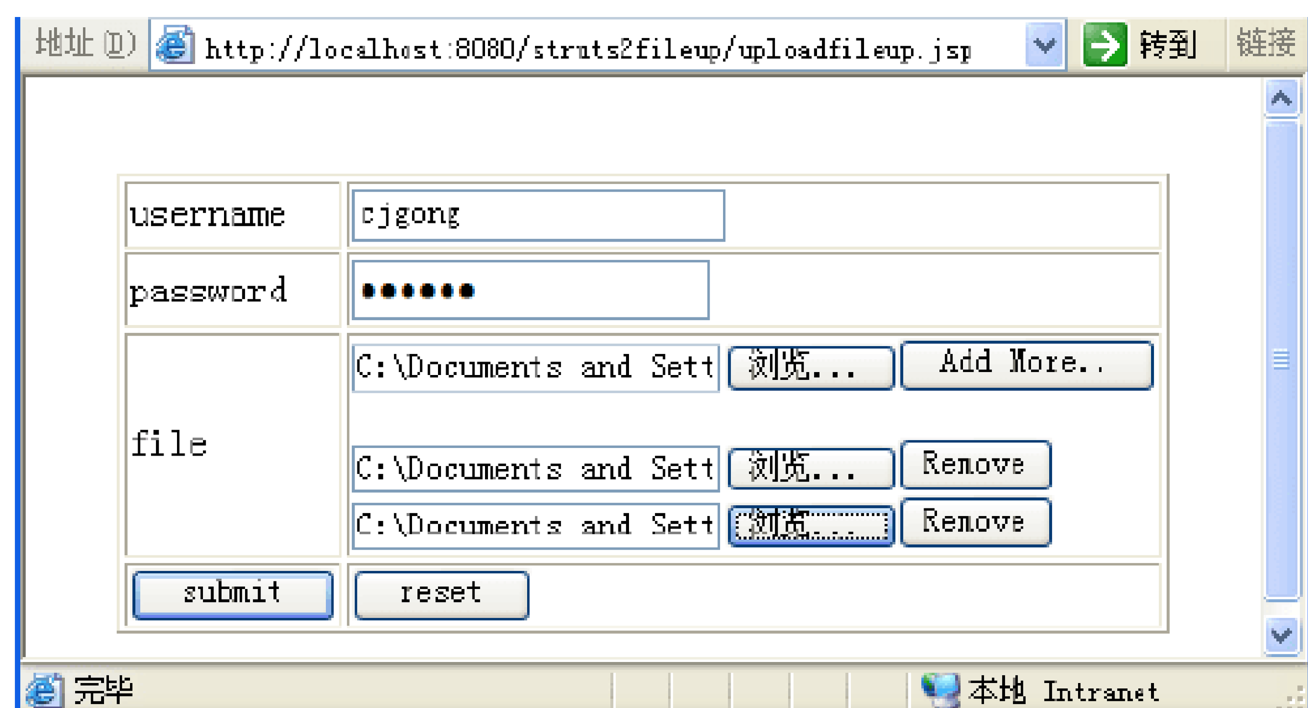


图 15.5　选择三个文件

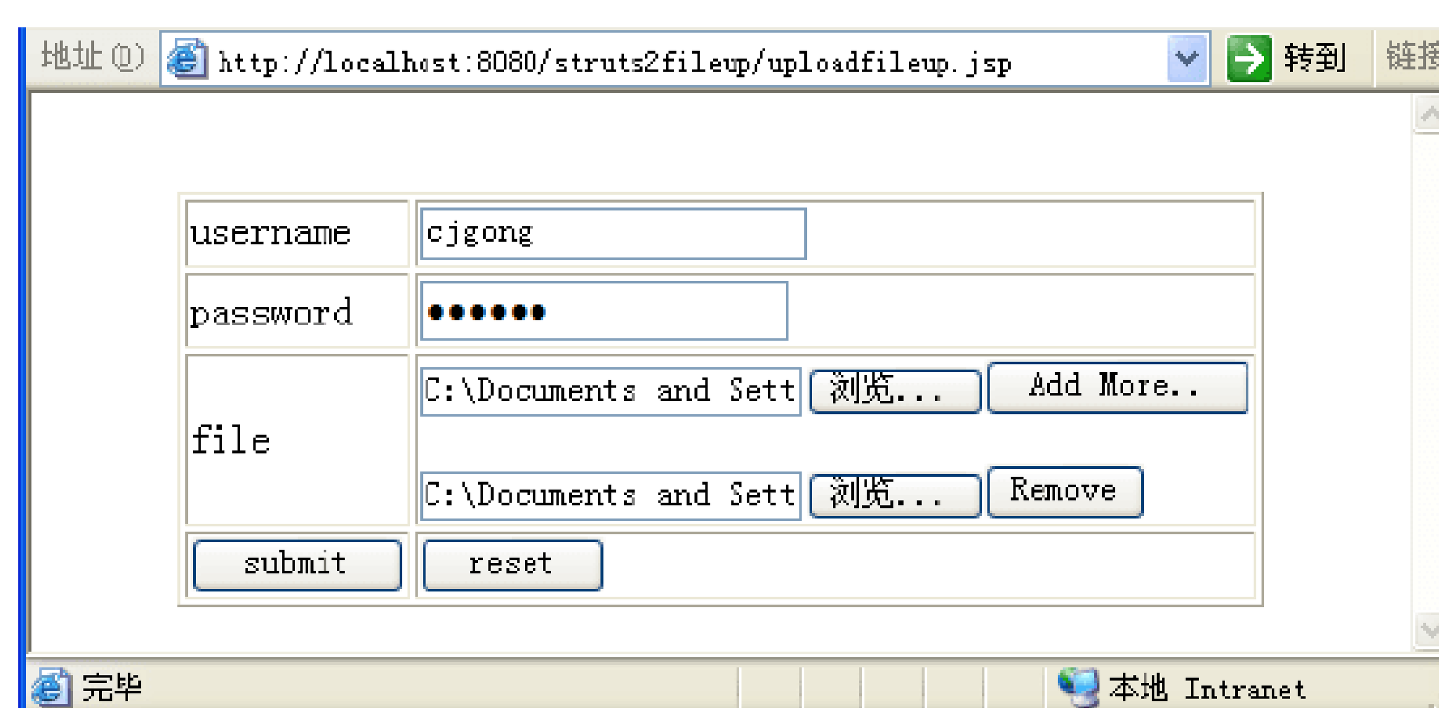


图 15.6　删除一个 file 域

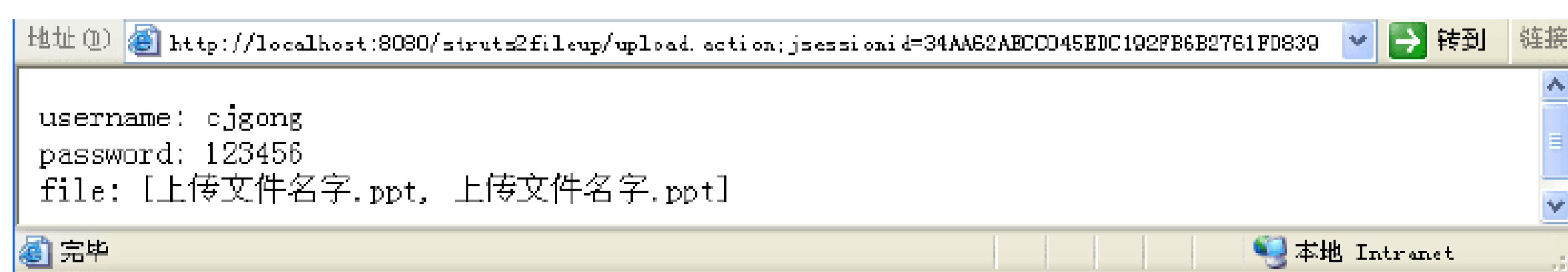


图 15.7　显示上传文件信息

3．文件上传失败

当浏览者在 uploadfileup.jsp 网页中选择的文件太大时，单击 submit 按钮就会出现如图

15.8 所示的页面。如果浏览者在 uploadfileup.jsp 网页中选择的文件类型不是 ppt 时，单击 submit 按钮就会出现如图 15.9 所示的页面。



图 15.8　文件太大错误



图 15.9　文件类型出错

4．下载文件

浏览者可以通过 download.jsp 网页（如图 15.10 所示）来实现文件的下载，当单击 download 链接时就可以打开“文件下载”对话框（如图 15.11 所示）。在该对话框中只要单击“保存”按钮就可以出现“另存为”对话框（如图 15.12 所示），在该对话框中单击“保存”按钮就会把所需下载的文件下载到相应的文件夹目录中。



图 15.10　下载页面

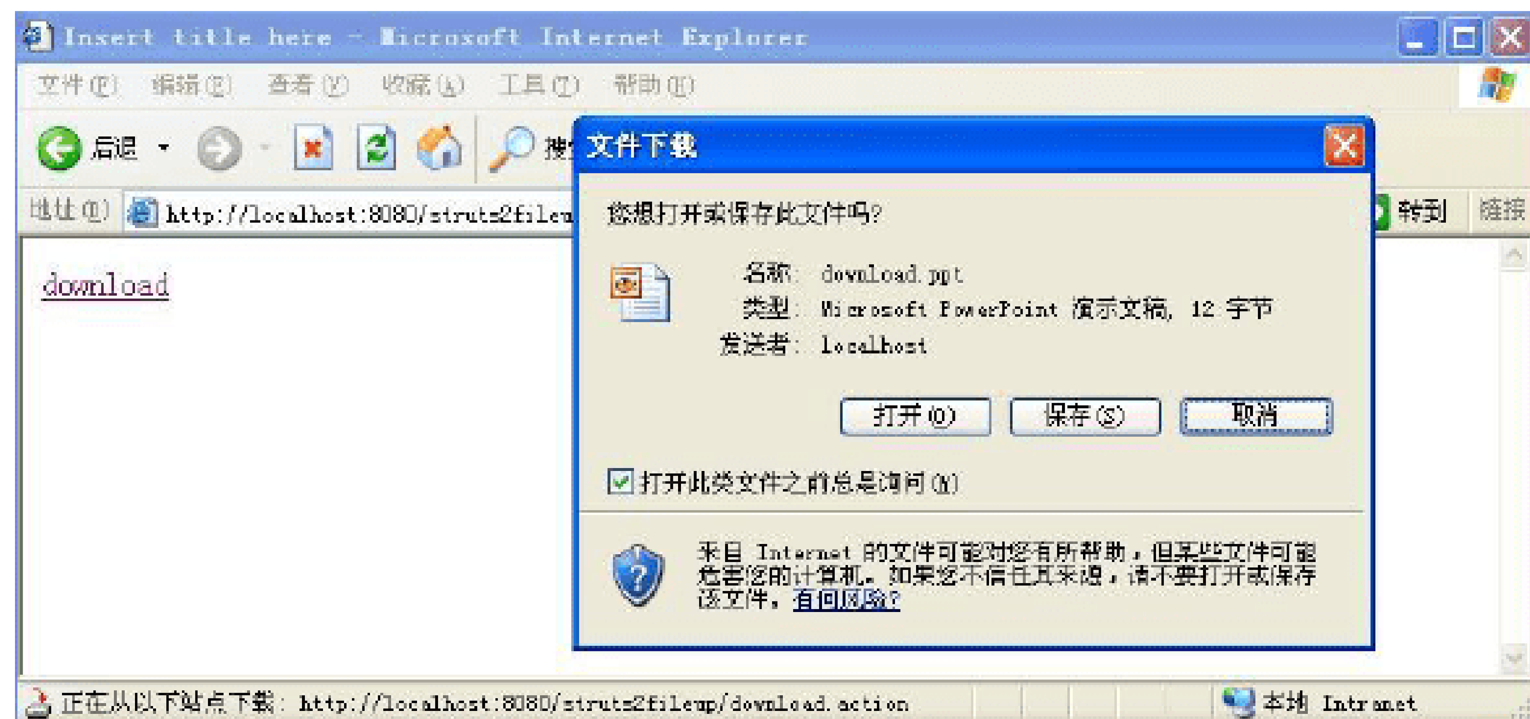


图 15.11　文件下载对话框

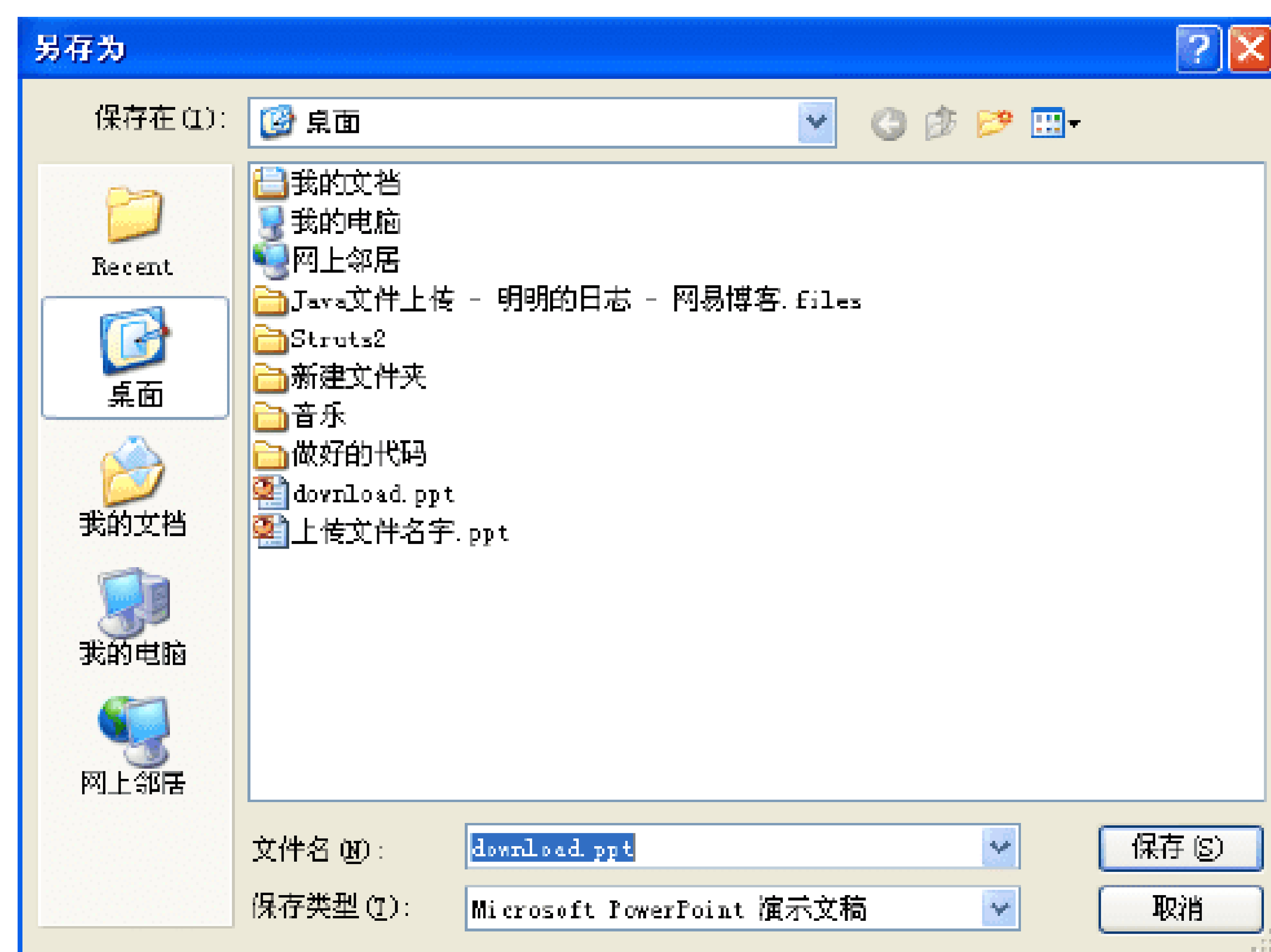


图 15.12　另存为对话框

15.2　文件上传组件 FileUpload

Components-FileUpload 组件是 Apache 组织的开源项目之一，用来实现 Java Web 环境中的文件上传功能，该组件不仅可以实现单文件的上传功能还能实现多文件的上传功能。由于组件 FileUpload 依赖于 Commons-IO 组件，所以在具体开发时除了要下载 Components-FileUpload 组件外，还必须下载 Commons-IO 组件。

15.2.1　下载文件上传组件（FileUpload）

Components-FileUpload 组件是 Jakarta 项目组开发的一个功能非常强大的上传文件组件，该组件主要用来实现上传功能，目前最新的版本为 FileUpload 1.2.1，具体的下载步骤如下。

（1）首先访问 apache 组织的官方网站（http://www.apache.org/），如图 15.13 所示。在该页面中单击 Foundation 导航栏就可以进入 apache 组织的产品列表。

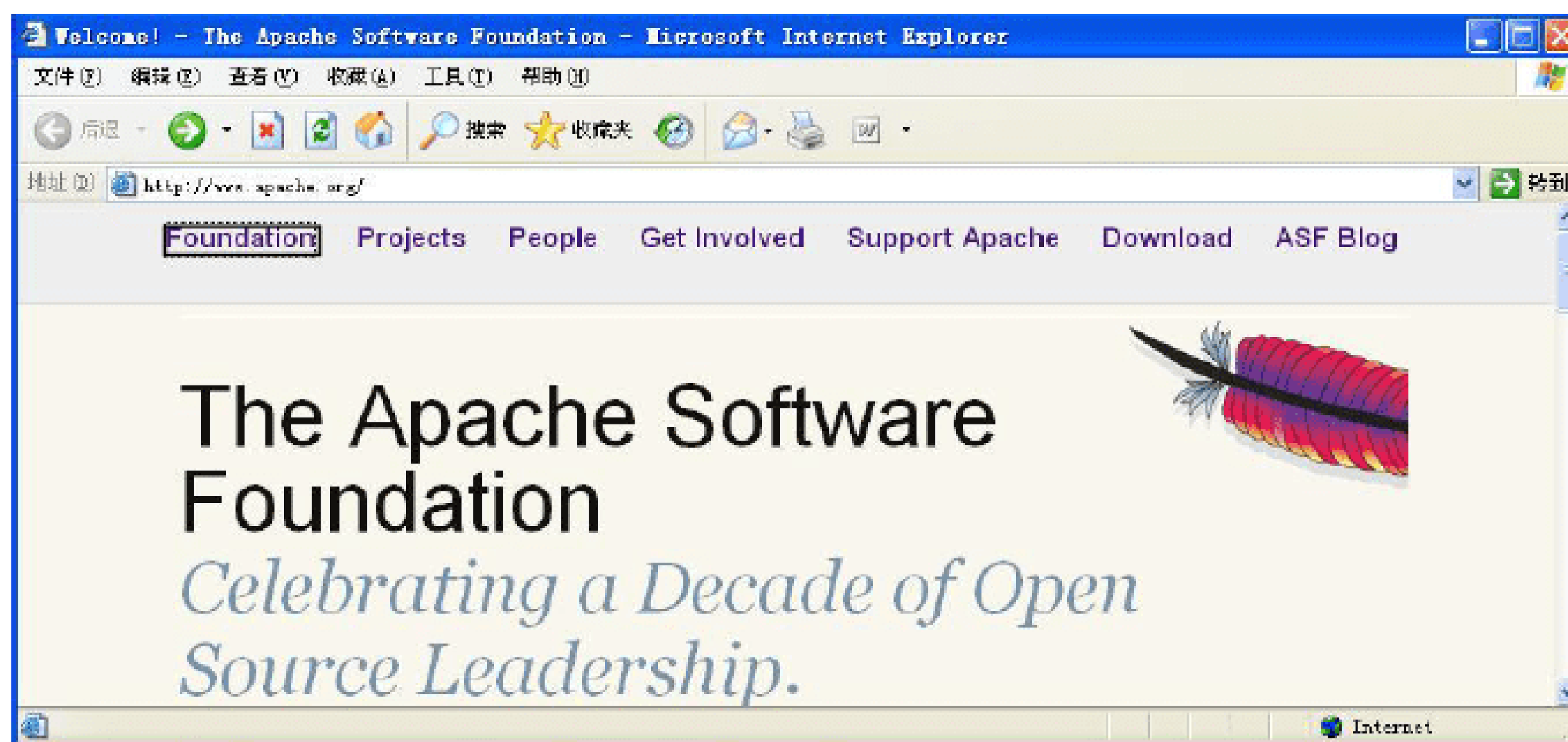


图 15.13　apache 官方首页

（2）在关于 apache 组织的产品列表页面（如图 15.14 所示）后，选择右边 Apache Projects 栏目中的 Jakarta 选项，就可以打开关于 Jakarta 产品的页面，如图 15.15 所示。在该页面中单击左边的 Commons 选项就可以直接进入关于 Commons 的页面。

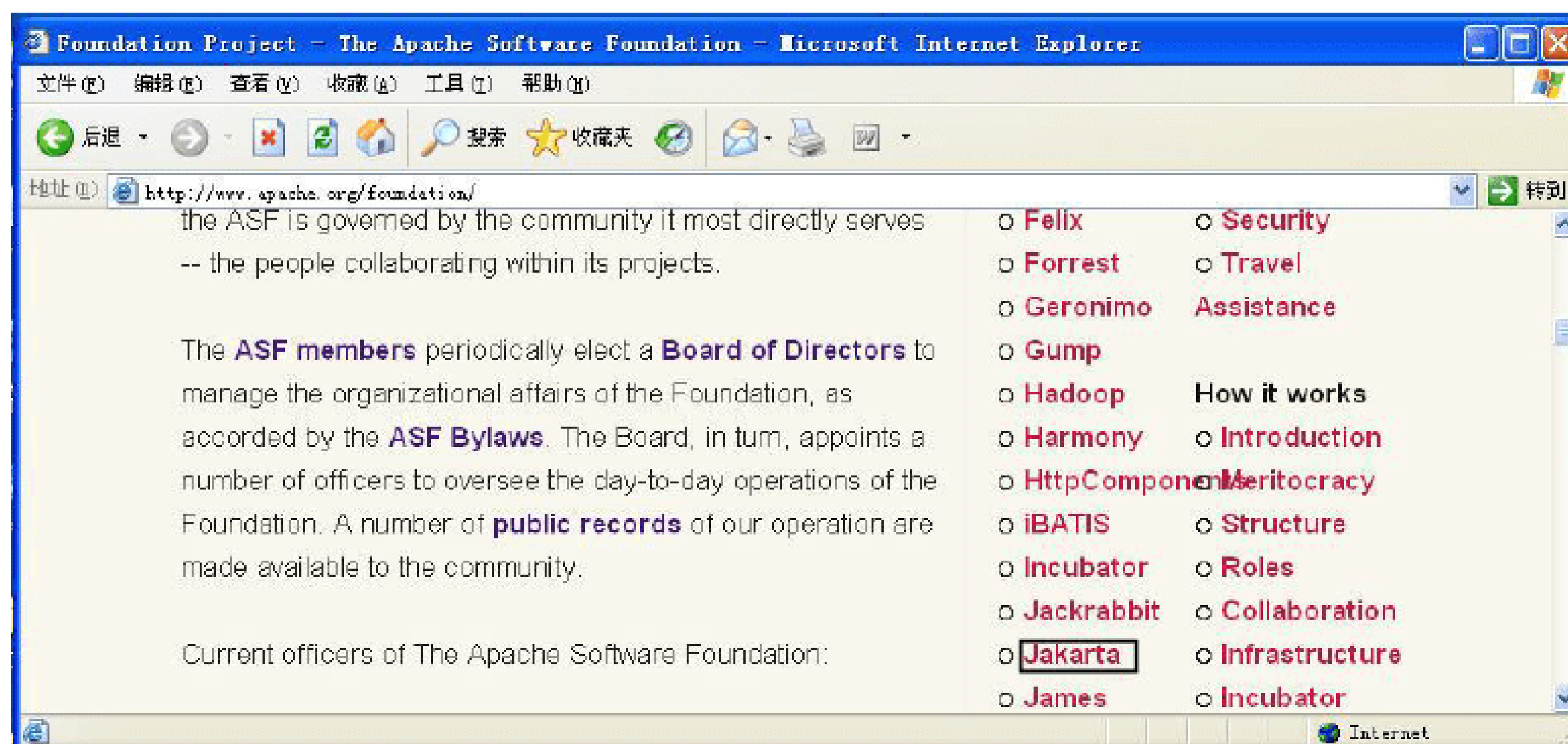


图 15.14　Jakarta 产品页面

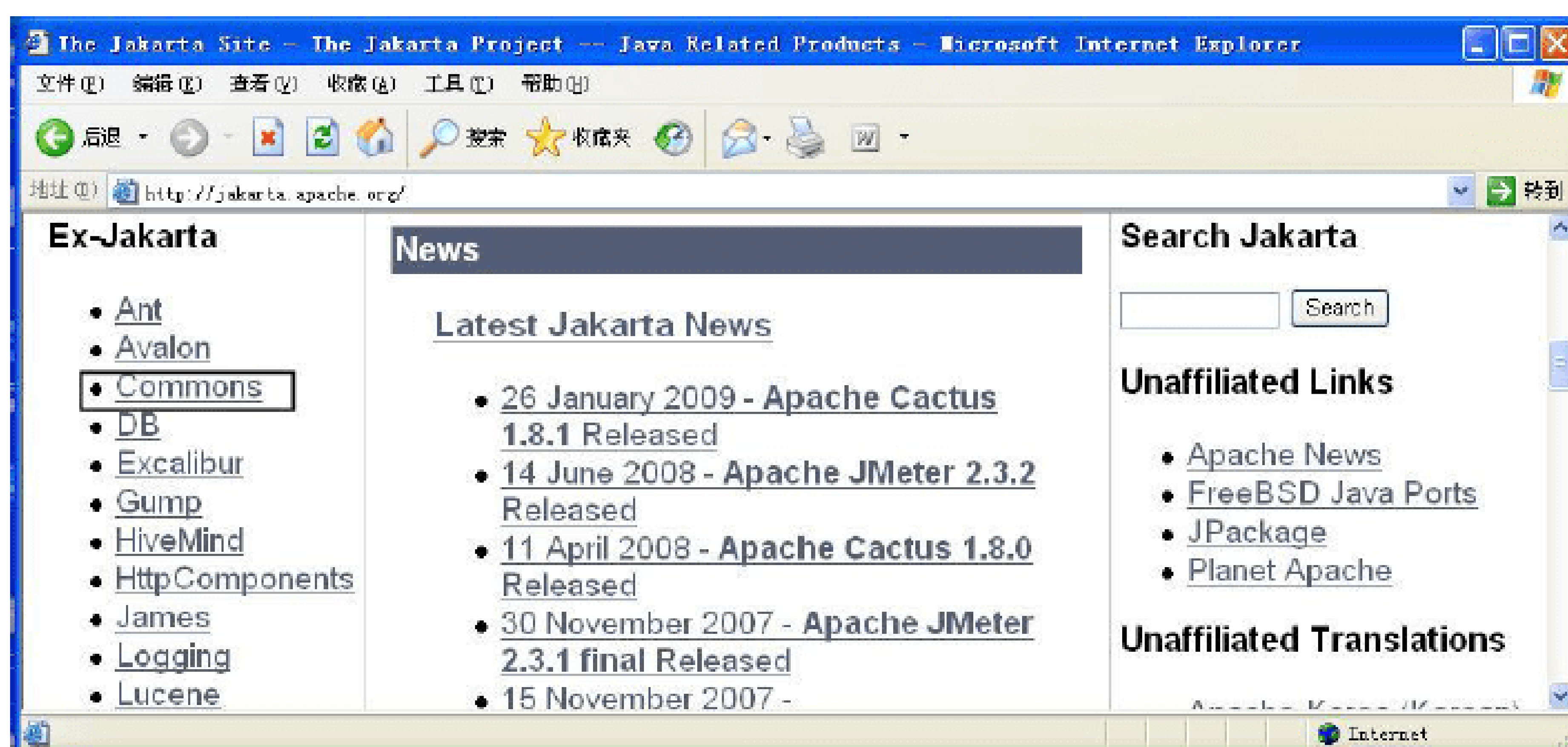


图 15.15　关于 Commons 的页面

（3）在关于 Commons 的页面（如图 15.16 所示）中，从 Components 目录中选择 FileUpload 和 IO 两个 jar 包来下载。

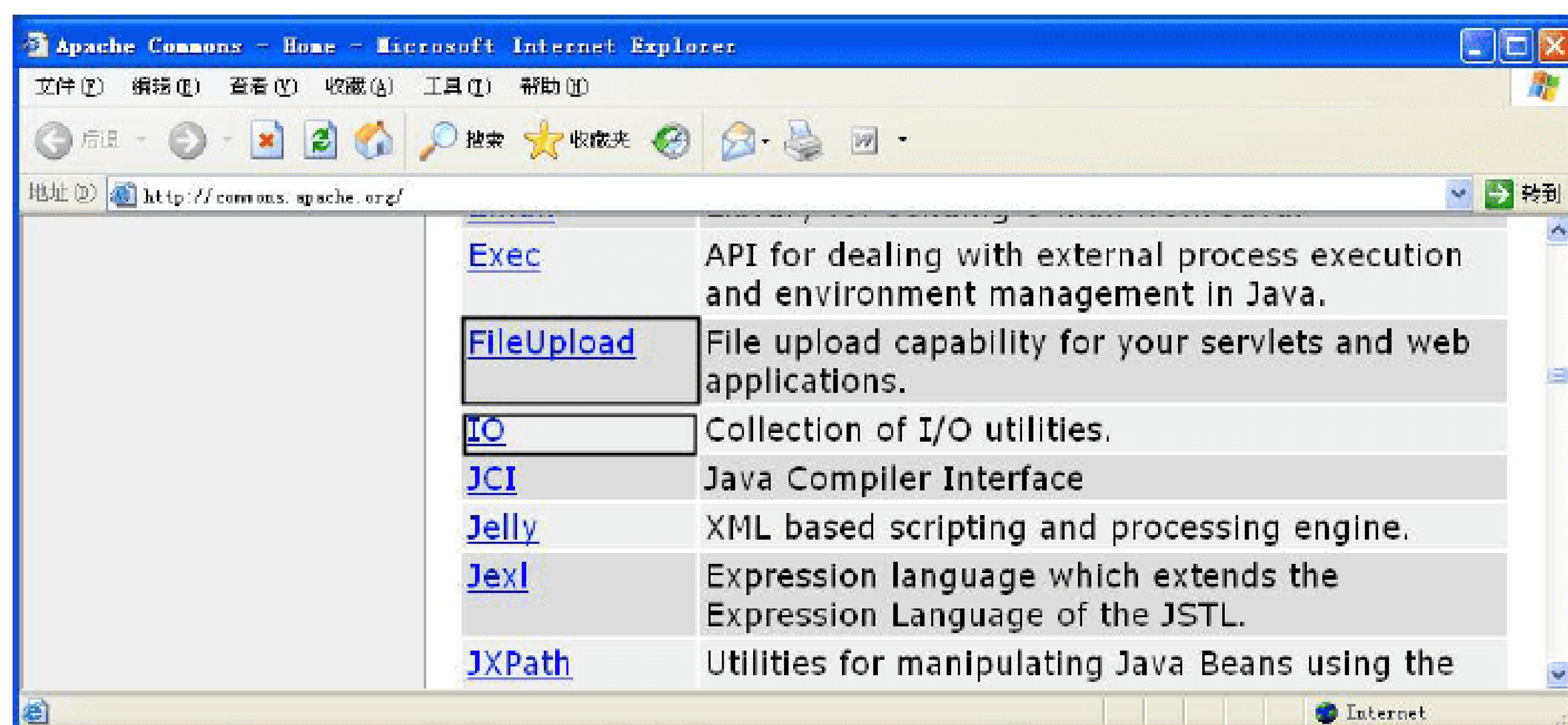


图 15.16　选择相应组件

（4）当单击 FileUpload 链接后，就可以转到关于 FileUpload 组件的页面（如图 15.17 所示）。在该图中的 Full Releases 选项中单击 FileUpload 1.2.1 下面的 here 链接就可以转到下载 FileUpload 组件的页面。

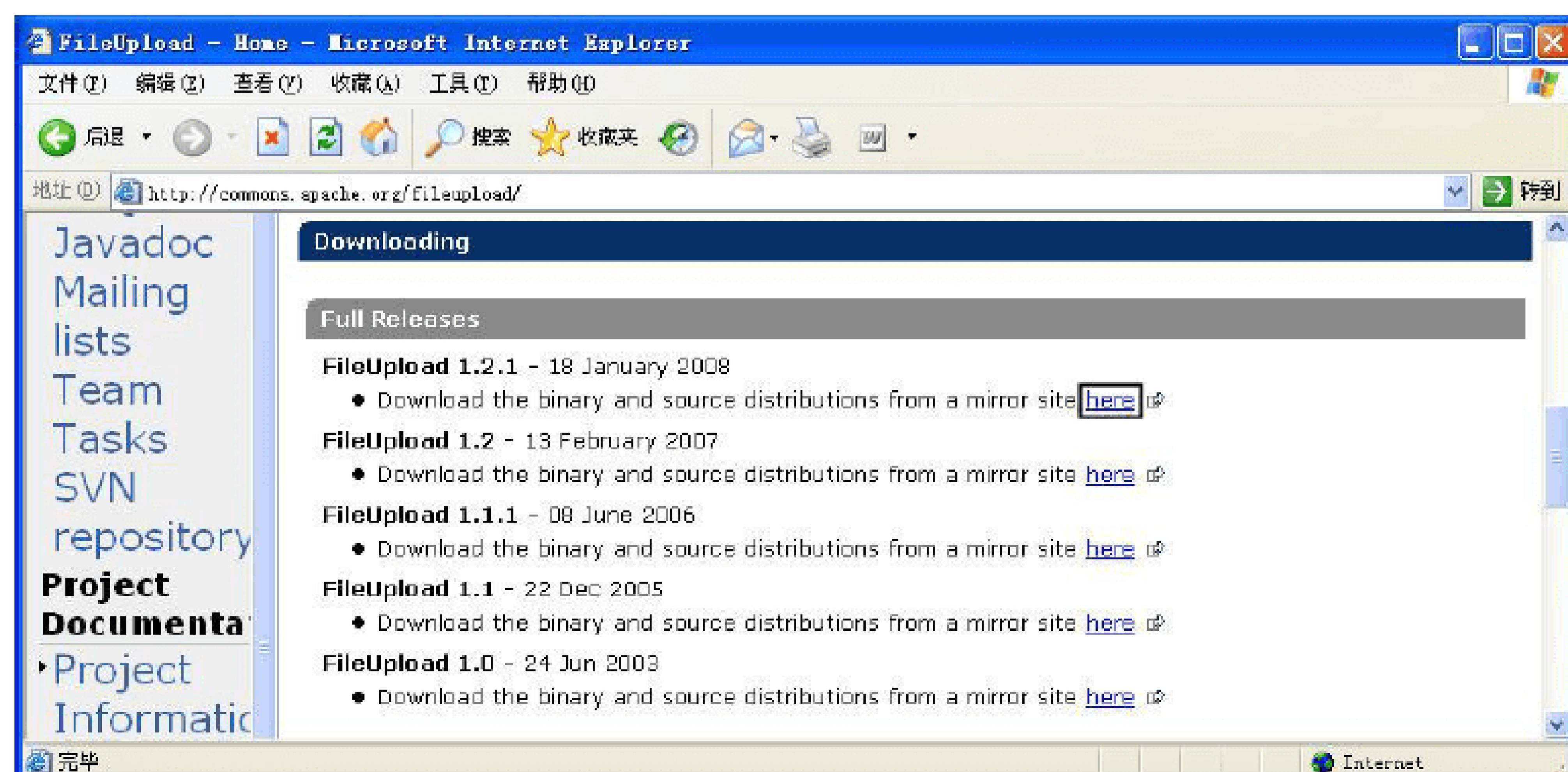


图 15.17　关于 FileUpload 页面

（5）在关于下载 FileUpload 组件的页面中，从 Binary 目录中选择 1.2.1.zip 版本来完成该组件的下载，如图 15.18 所示。

（6）当在图 15.16 中单击 IO 链接后，就可以转到关于 IO 组件的页面（如图 15.19 所示）。在该图中的 Releases 选项中单击 Download now 链接就可以转到下载 IO 组件的页面。

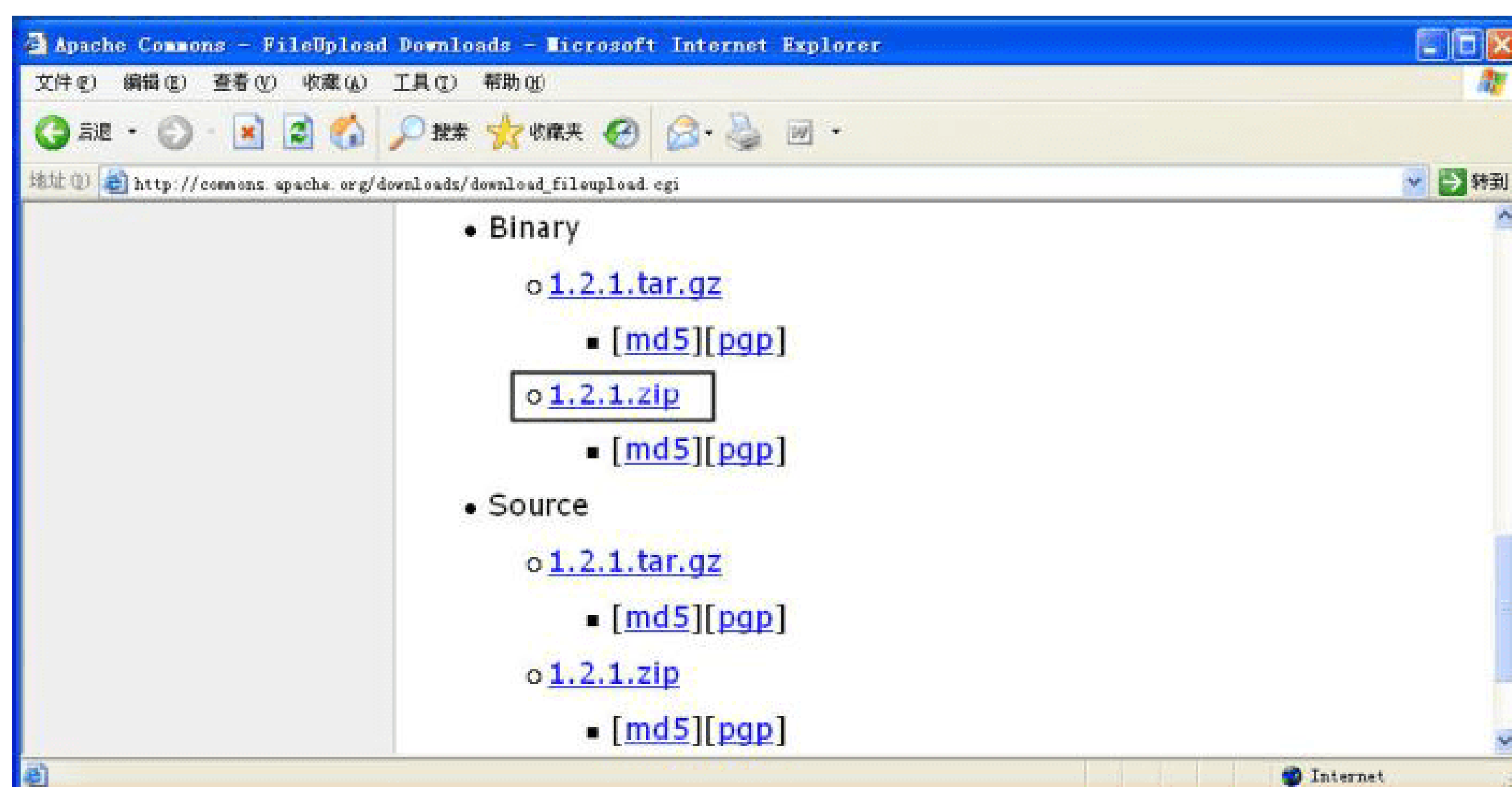


图 15.18　下载 FileUpload 组件的页面

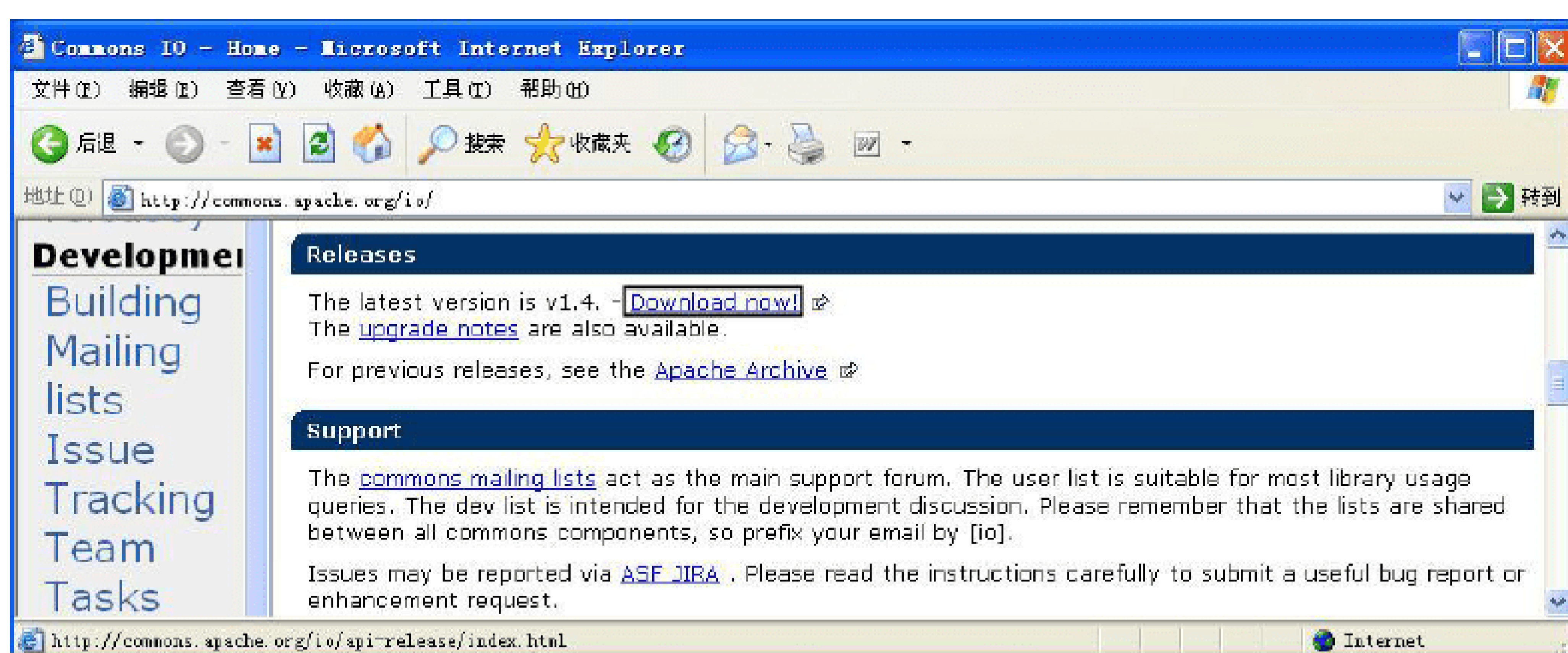


图 15.19　关于 IO 组件的页面

（7）在关于下载 IO 组件的页面中（如图 15.20 所示），从 Binary 目录中选择 1.4.zip 版本来完成该组件的下载。

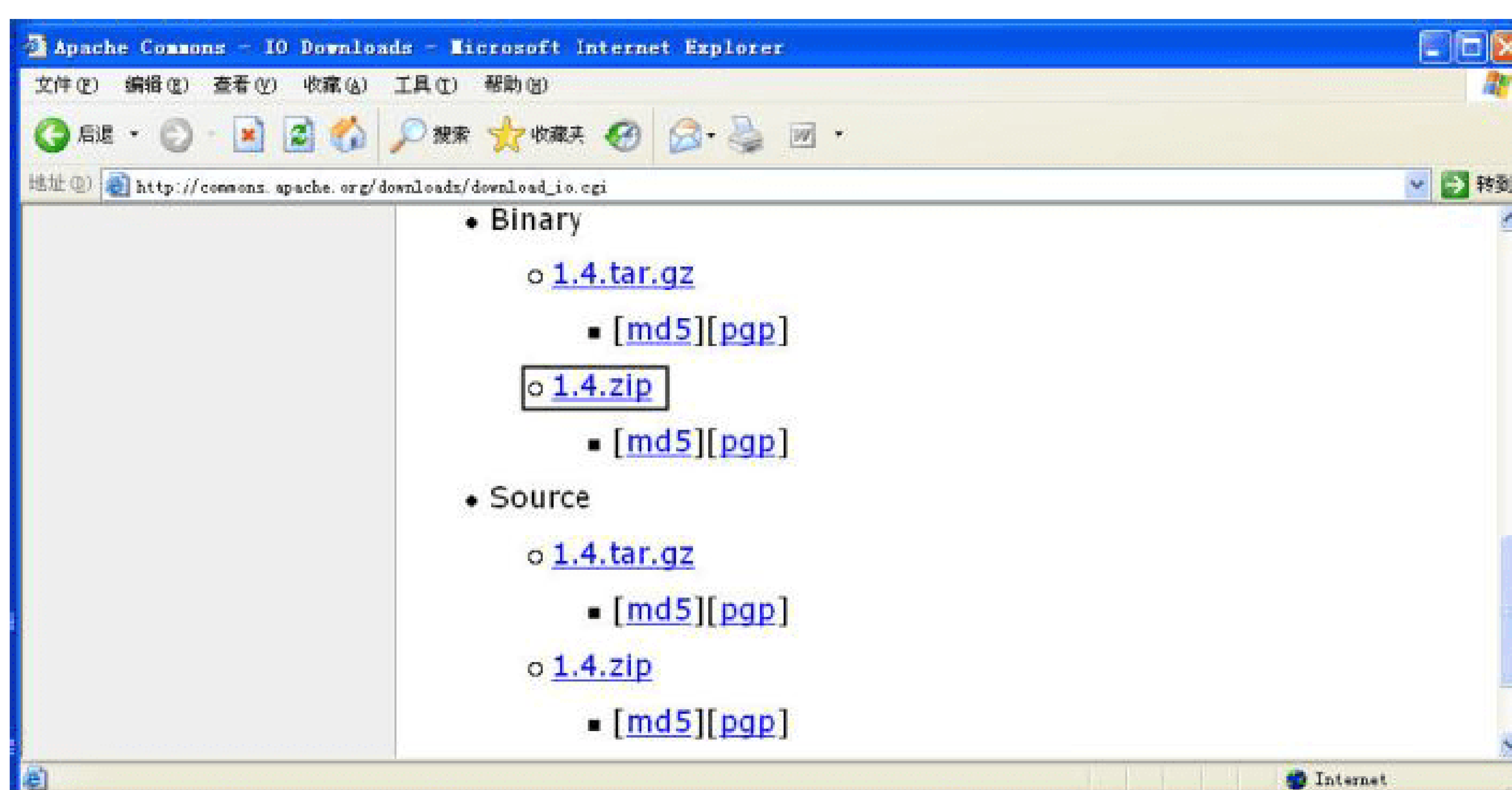


图 15.20　下载 IO 组件的页面

至此，就完成对 Components-FileUpload 和 Components-IO 组件的下载。

15.2.2　了解和管理 Components-FileUpload 组件及其相关组件

15.2.1 节介绍了如何下载 Components-FileUpload 组件和相关组件，下载完这些组件后就可以在 Java Web 项目中使用该框架。在具体使用之前，先解压 commons-fileupload-1.2.1-bin.zip 文件，该压缩包目录 commons-fileupload-1.2.1-bin.zip\commons- fileupload-1.2.1\

lib，如图 15.21 所示。

在上述目录中 3 个文件构成了 Components-FileUpload 组件的架包，这 3 个文件如下所示。

- ❑ commons-fileupload-1.2.1.jar：关于 FileUpload 组件的核心架包。
- ❑ commons-fileupload-1.2.1-sources.jar：关于 FileUpload 组件的源代码。
- ❑ commons-fileupload-1.2.1-javadoc.jar：关于 FileUpload 组件的帮助文档。

为了便于使用，可以复制 commons-fileupload-1.2.1\lib 文件下的 commons-fileupload-1.2.1.jar 这个 jar 文件到相应的文件夹中，然后把这个 jar 文件在 MyEclipse 开发环境中专门建立一个名为 fileupload 用户库，如图 15.22 所示。

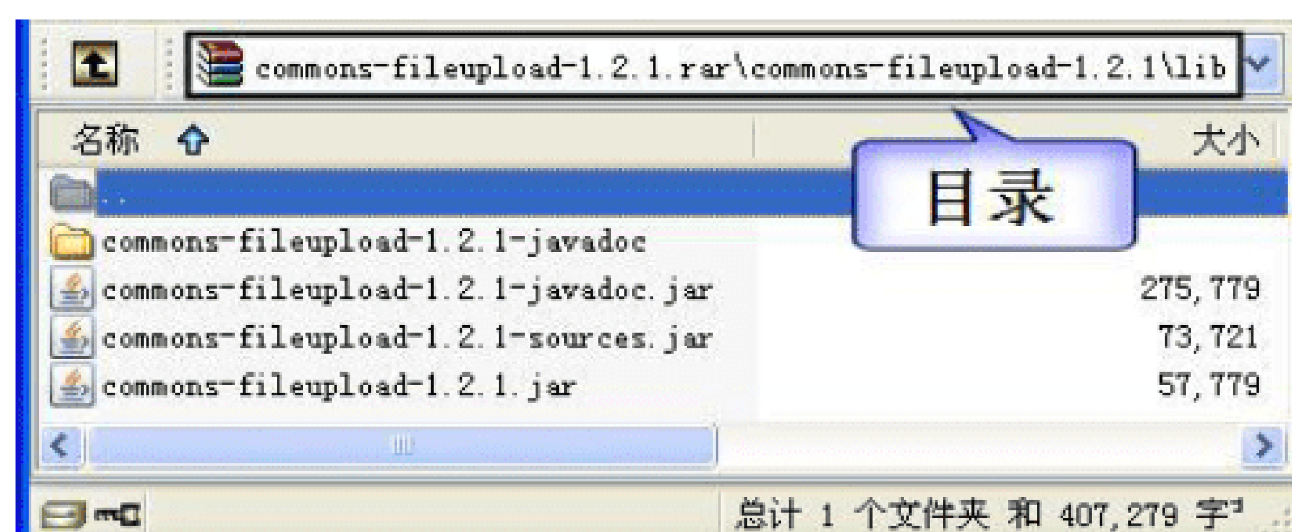


图 15.21　目录结构

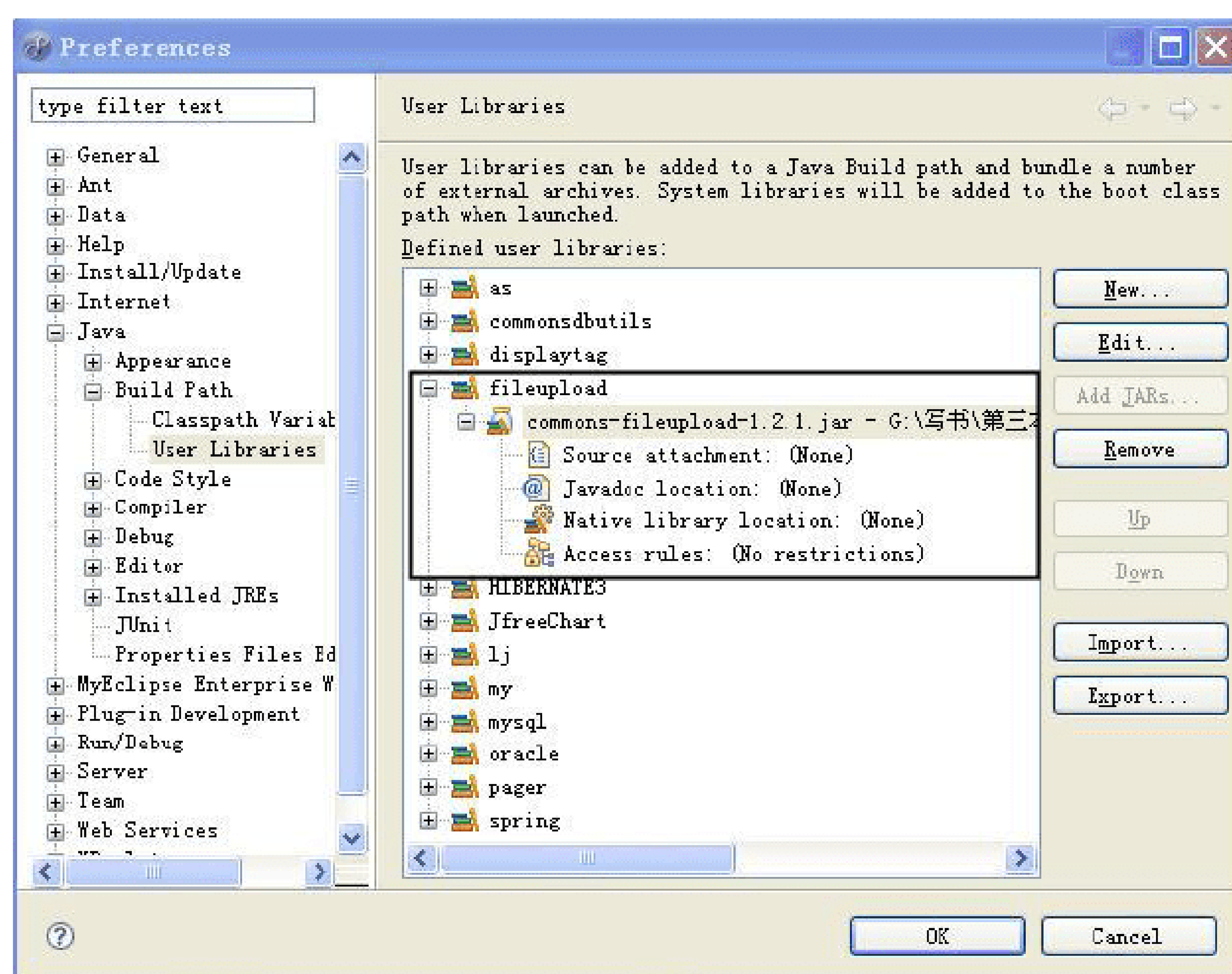


图 15.22　fileupload 用户库

在具体开发关于 Components-FileUpload 组件的程序时，会经常查看关于 Components-FileUpload 组件的帮助文档，可以通过解压 commons-fileupload-1.2.1-javadoc.jar 文件，然后单击 index.html 文件打开帮助文档的首页，如图 15.23 所示。

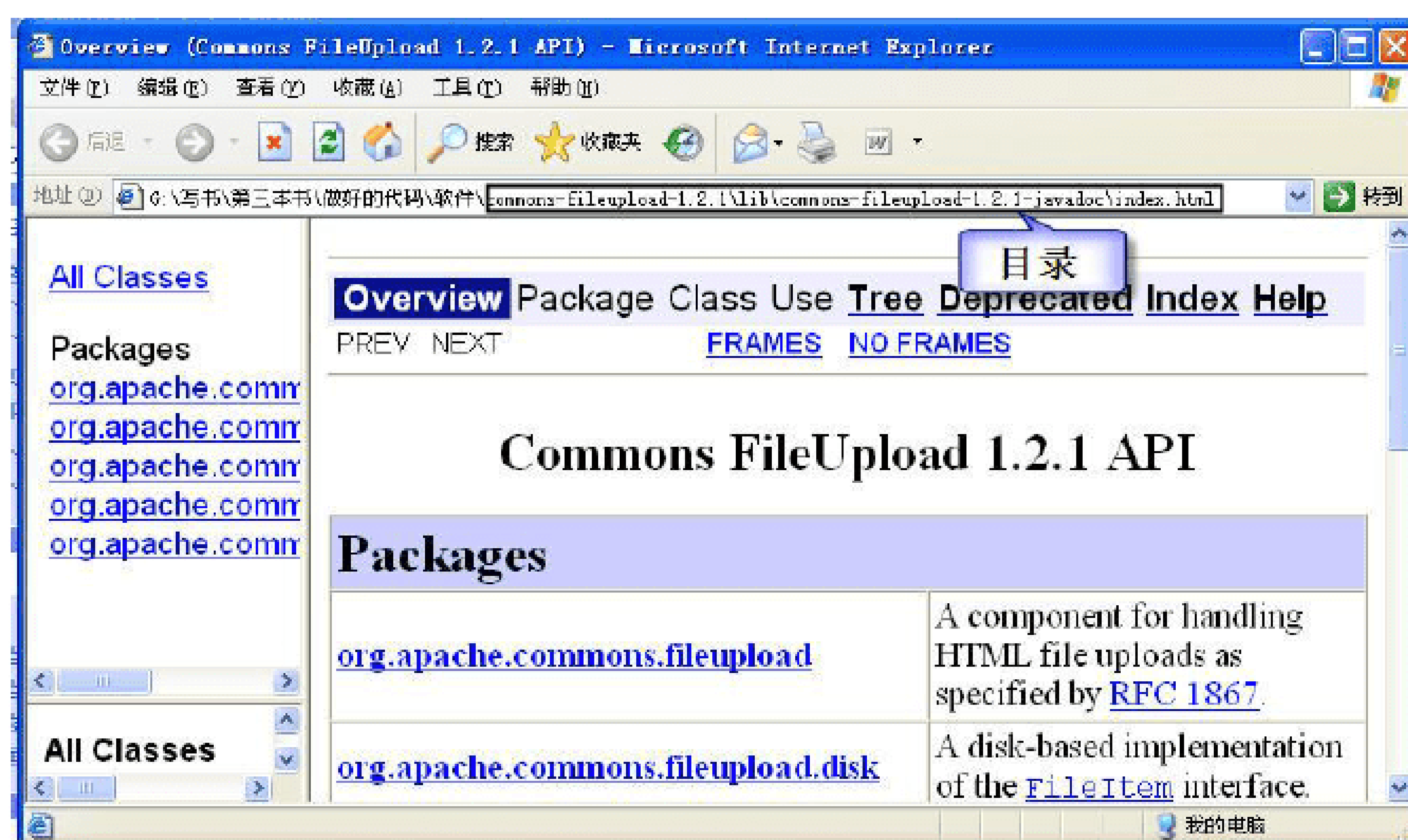


图 15.23　关于 FileUpload 组件帮助文档首页

对于 Components-IO 组件的目录结构和帮助文档首页，与 Components-FileUpload 组件一样，分别如图 15.24 和图 15.25 所示。然后接着为 fileupload 用户库添加关于 Components-IO

组件架包，如图 15.26 所示。

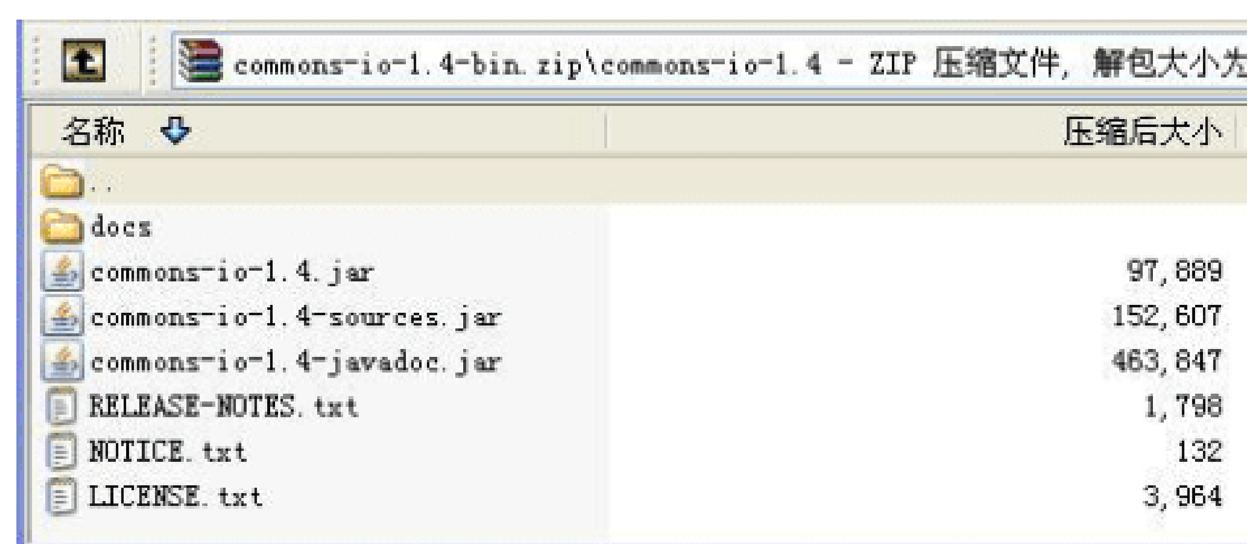


图 15.24　关于 IO 组件目录结构

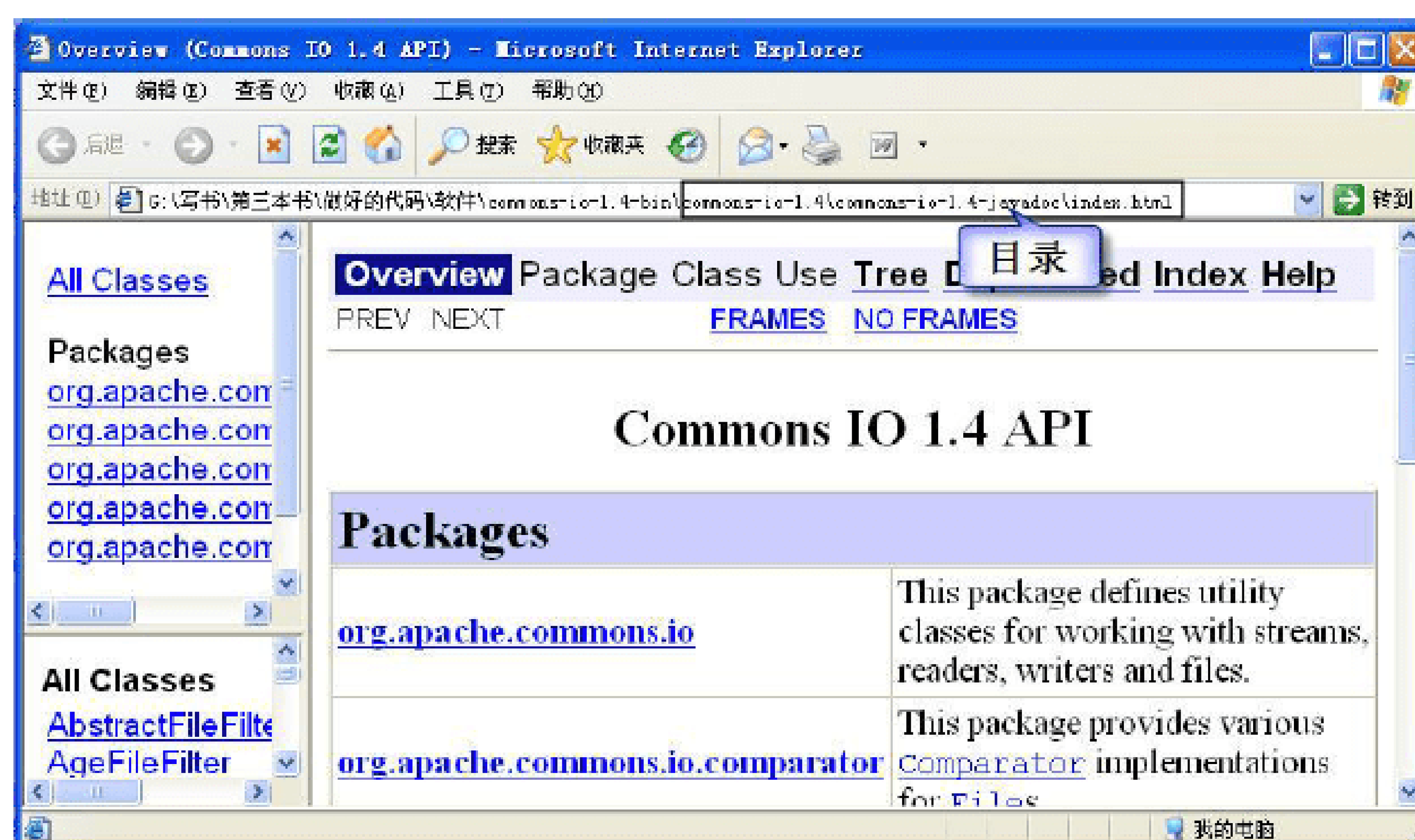


图 15.25　关于 IO 组件帮助文档首页

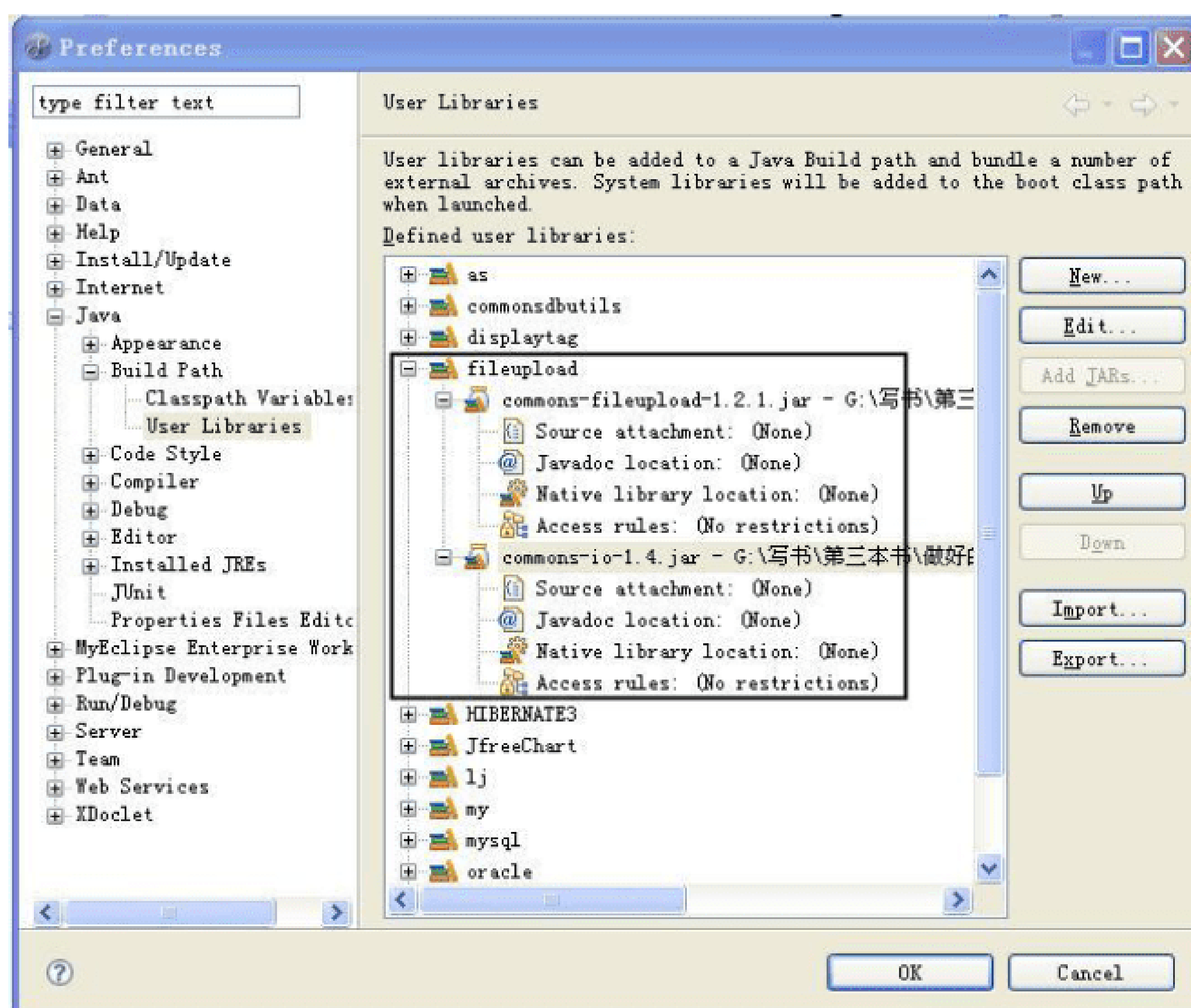


图 15.26　fileupload 用户库

至此，就完成对 Components-FileUpload 组件及相关组件在 MyEclipse 开发环境的配置。

15.3　初步使用文件上传组件（Components-FileUpload）

在使用已经开发好各种文件上传组件时，虽然底层实现各不相同，但是在具体开发使用时却没有太大的区别。对于好的组件来说，如果对于使用者来说简单易用，那么对于自己的底层实现上却非常复杂。所以该节将简单介绍一下实现文件上传功能的原理，该程序流程如图 15.27 所示。

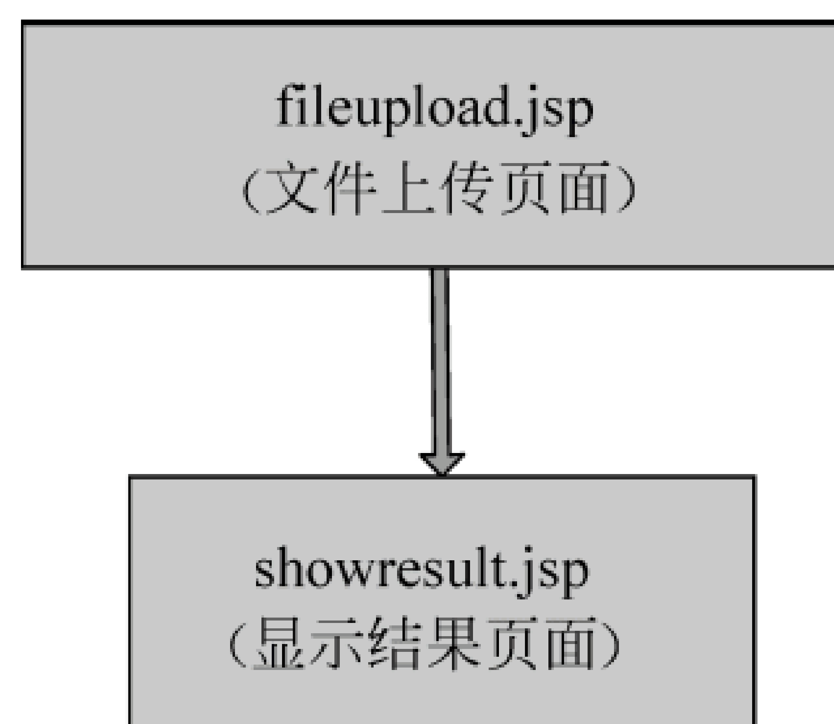


图 15.27　程序关系图

（1）首先使用者在 fileupload.jsp 页面上选择所要上传的文件，以及填写对该文件的描述。

（2）当使用者填写完相应的信息后，经过相应的处理，则会转到 showresult.jsp 页面，然后在该页面中显示出所要上传文件的相关内容。

15.3.1　选择所要上传的文件

fileupload 页面用来让使用者实现选择所要上传的文件。在该页面中使用者不仅需要选择所要上传的文件，而且还必须填写该文件的相关描述。代码 15.1 用来获取所要上传的文件。

代码 15.1　选择上传文件：fileupload.jsp

```
...
<head>
<meta http-equiv="Content-Type" content="text/html; charset=GB18030">
<title>fileup</title>                              <!--页面标题-->
</head>
<body>
    <form  action="showresult.jsp"  method="post"  enctype="multipart/
    form-data">
    Information: <input type="text" name="info"><br><!--文件描述文本框-->
    File: <input type="file" name="file"><br> <!--文件上传控件-->
    <input type="submit" name="submit" value=" submit ">
    </form>
</body>
...
```

【代码解析】

- 在该项目中如果要想实现文件的上传，首先要在 fileupload.jsp 页面中利用表单把所要上传的数据编码方式设置为二进制数据方式。
- 如果想实现文件上传功能，必须设置表单元素<form>中属性 method 的值为 post 而不能是 get。
- 在表单元素<form>中有一个属性 enctype 用来设置上传数据的编码方式，其有 3 个值，具体含义如表 15.1 所示。

表 15.1　enctype的属性值

属　性　值	应 用 范 围
text/plain	该种方式主要用于电子邮件方面的应用
multipart/form-data	该种方式主要用于上传文件等的应用，当利用该种方式上传文件时，首先会把数据转化成二进制数据，然后才会进行上传
application/x-www-form-urlencoded	该种方式主要用于只要是能输出网页的应用都可以，所以该值为默认值，不过当传送的内容包含大量的非 ASCII 字符的文本或二进制数据时，效率比较低

从上述表中可以看出，只有把属性 enctype 的值设置为 multipart/form-data，就能实现完整的传递文件数据。

15.3.2　显示上传文件

showresult.jsp 页面用来显示所上传文件的相关内容，其不仅会显示出该文件的描述信

息、文件名信息，同时还会显示出该文件的内容。代码 15.2 实现了显示上传文件的相关信息。

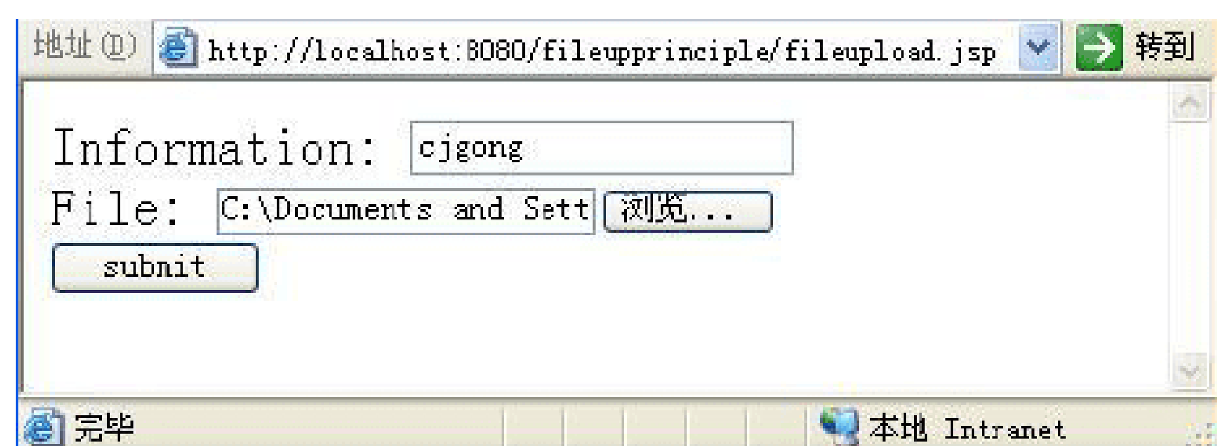
代码 15.2　选择上传文件：showresult.jsp

```
...
<%@ page import="java.io.*" %>                         <!--导入 IO 包-->
...
<body>
<%
    InputStream is = request.getInputStream();          //获取字节输入流
    //创建带有缓冲功能字符流
    BufferedReader br = new BufferedReader(new InputStreamReader(is));
    String buffer = null;
    while((buffer = br.readLine()) != null)             //遍历输入流 br
    {
        out.print(buffer + "<br>");                     //输出字符 buffer
    }
%>
</body>
...
```

【代码解析】

在上述代码中首先通过 request.getInputStream()方法获取由 fileupload.jsp 页面传送的二进制数据流，然后通过类 InputStreamReader 封装该输入流为字符流，同时通过类 BufferedReader 使该输入流带有缓冲功能，最后遍历该输入流中的字符并输出在相应页面上。

单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/fileupprinciple/fileupload.jsp 就会打开选择上传文件页面，在该页面中输入相应的信息（如图 15.28 所示），单击 submit 按钮就会实现转到如图 15.29 所示的显示上传文件相关信息的页面。所要上传的文件如图 15.30 所示。



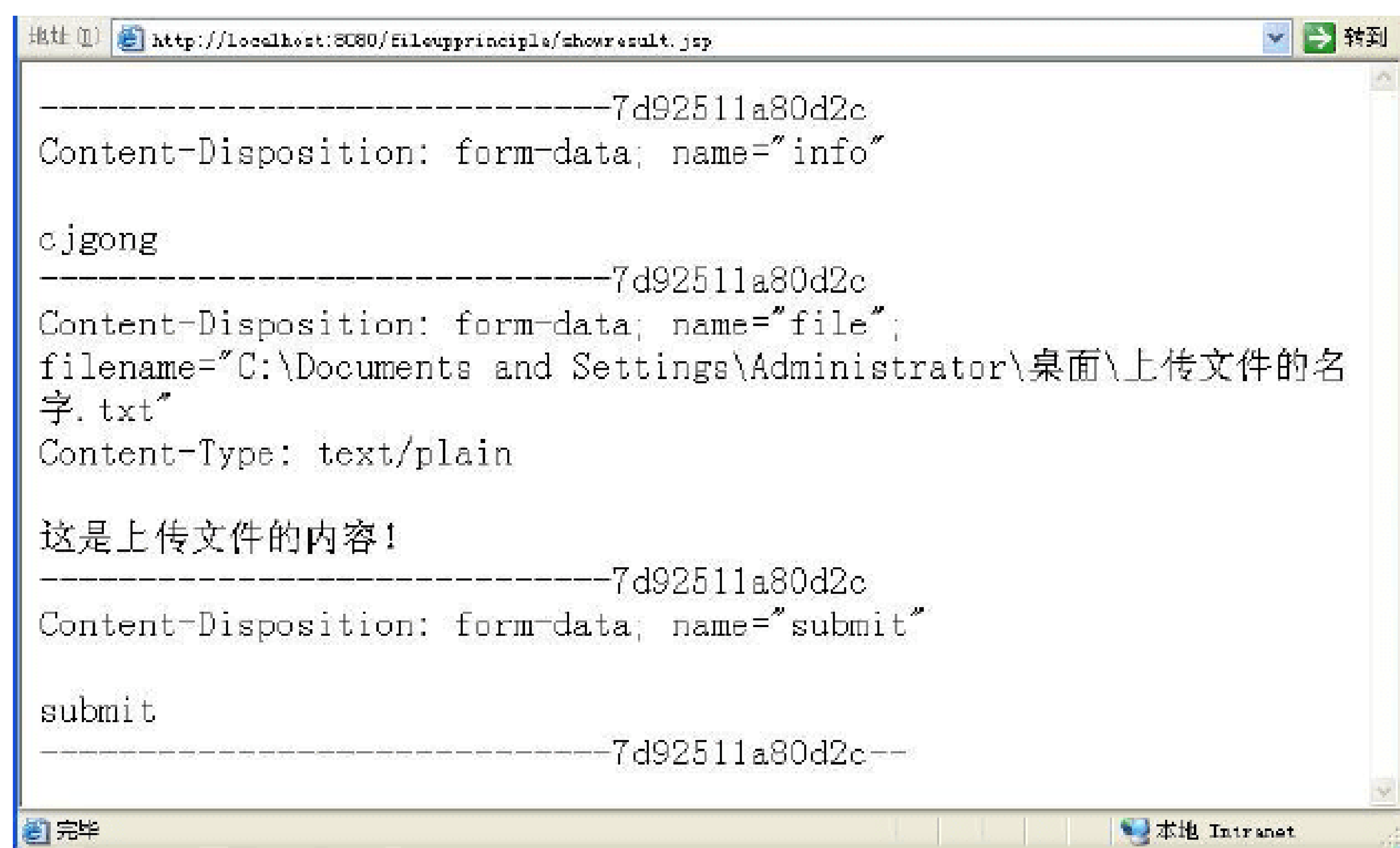


图 15.28　输入相应内容　　　　图 15.29　显示上传文件的相关信息

注意： 显示上传文件的相关信息页面中之所以会出现“——————7d92511a80d2c”字符串，因为该字符串是关于 HTTP 协议请求头的相关信息。

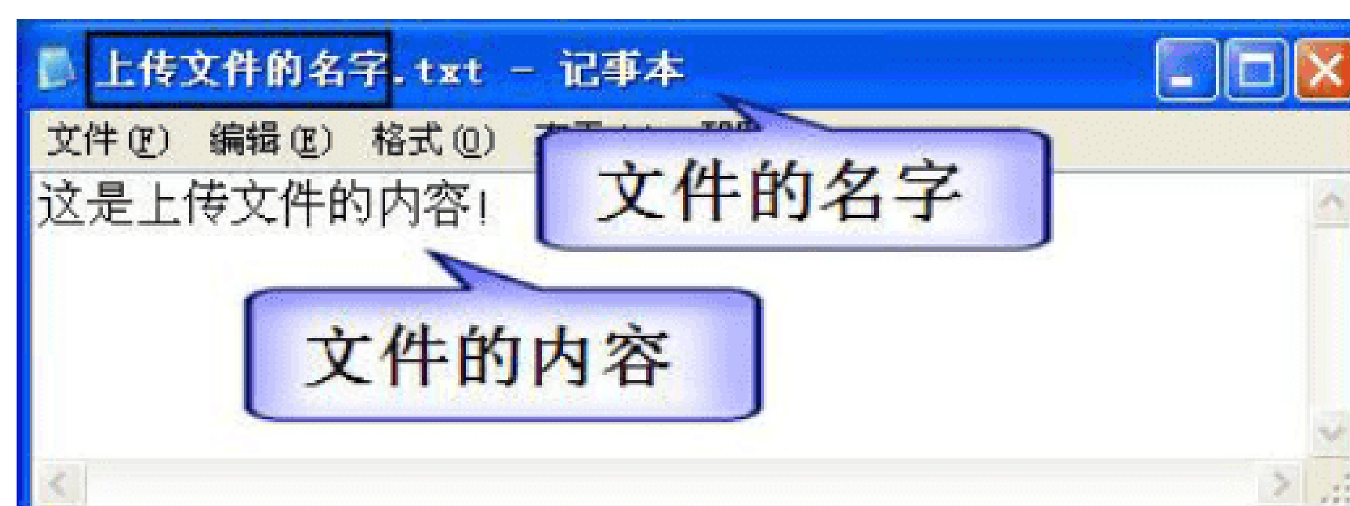


图 15.30　所要上传的文件

通过 fileupprinciple 项目可以知道，首先通过表单 form 把关于文件的二进制数据传送给服务器，然后在服务器端获取该二进制数据，最后在服务器端存储这些数据实现文件的上传功能。

15.4　单文件的上传

本节通过 Struts 2.x+Components-FileUpload 框架技术来实现单文件上传功能，只有具有一定权限的用户才能进行文件的上传，本节为了便于讲解关于用户权限的功能就不具体实现。

15.4.1　选择所要上传的文件

fileupload 页面用来让使用者实现选择所要上传的文件。只有具有一定权限的用户才能实现文件上传，所以在该页面中不仅要选择上传的文件，还必须填写用户名和密码，具体内容如代码 15.3 所示。

代码 15.3　选择所要上传的文件：upload.jsp

```
...
<%@ taglib prefix="s" uri="/struts-tags" %>   <!--引入 struts2.x 的标签库-->
...
<body>
        <table align="center" width="50%">
            <td>
                <s:fielderror cssStyle="color:red" />  <!--显示错误信息-->
            </td>
        </table>
        <!--表单-->
        <s:form action="upload" theme="simple" enctype="multipart/
        form-data">
            <table">
...
                    <td>                                <!--用户名输入框-->
                        username
                    </td>
                    <td>
                        <s:textfield name="username"></s:textfield>
                    </td>
                    <td>                                <!--密码输入框-->
                        password
                    </td>
```

```
                    <td>
                        <s:password name="password"></s:password>
                    </td>
                    <td>                                <!--上传文件框-->
                        file
                    </td>
                    <td >
                        <s:file name="file"></s:file>
                    </td>
                    <td>                                <!--确认按钮-->
                        <s:submit value=" submit "></s:submit>
                    </td>
                    <td>                                <!--重置按钮-->
                        <s:reset value=" reset "></s:reset>
                    </td>
...
            </table>
        </s:form>
</body>
...
```

注意：上述代码中通过 Struts 2.x 框架的文件上传元素 file 来实现选择文件的功能，显示效果如图 15.31 所示。



图 15.31　上传文件页面

15.4.2　实现文件的上传功能

实现文件的上传功能，首先通过输入流要获取 upload.jsp 页面传送过来的数据流信息，然后再通过输出流把文件信息存储到服务器的存储器中。代码 15.4 用来实现文件的上传功能。

代码 15.4　实现文件的上传：UploadAction.java

```
...
public class UploadAction extends ActionSupport
{
    private String username;                    //创建 username 属性
    private String password;                    //创建 password 属性
    private File file;                          //创建 file 属性
    private String fileFileName;                //创建 fileFileName 属性
    private String fileContentType;             //创建 fileContentType 属性
    //省略 username、password、file、fileFileName 和 fileContentType 属性
```

```
...
    public String execute() throws Exception        //编写 execute()方法
    {
        for (int i = 0; i < file.size(); ++i)
        {
            InputStream is = new FileInputStream(file.get(i));//创建输入流
            //创建父目录
            String root = ServletActionContext.getRequest().getRealPath(
                    "/upload");
            //创建输出流的目的文件
            File destFile = new File(root, this.getFileFileName());
            //创建输出流
            OutputStream os = new FileOutputStream(destFile);
            byte[] buffer = new byte[400];          //创建中间字节数组
            int length = 0;
            while ((length = is.read(buffer)) > 0)
                                                    //实现输入流到输出流的转换
            {
                os.write(buffer, 0, length);
            }
            is.close();                             //关闭输入流
            os.close();                             //关闭输出流
        }
        return SUCCESS;
    }
}
```

【代码解析】

- 在创建相应表单的上传文件元素 file 的属性时，其类型为 File，同时为了获取文件的其他信息，还创建了代表文件名称的 fileFileName 属性和代表文件类型的 fileContentType 属性。文件名称和文件类型的属性名字在 Struts 2.x 中是有规定的，即对于文件名称前面一部分必须跟对应文件域的属性 file 相同，而后一部分必须为 FileName 不能改变；对于文件类型前面一部分也必须跟对应文件域的属性 file 相同，而后一部分必须为 ContentType 不能改变。
- 在 execute()方法中，首先创建一个字节型输入流来读取传送过来的文件 file。接着通过 ServletActionContext.getRequest().getRealPath()获取存储文件的父目录，在该句代码中，ServletActionContext 表示关于 Servlet 的上下文，getRequest()方法获取 Request 对象，getRealPath("/upload")方法获取当前项目根目录下的 upload 文件夹。然后再创建一个 destFile 文件，该文件用来做为输出流的目的文件。最后再创建一个关于 destFile 文件的输出流。
- 当创建完输入流和输出流后，就可以创建一个中间字节数组 buffer 来实现输入流与输出流的转换，从而实现文件的上传功能。

15.4.3　关于 FileUpload 拦截器的配置

虽然在 15.4.2 节已基本完成上传文件的基本操作，但是如果想成功地实现该功能，还必须在 struts.xml 文件中对实现文件上传的 Action 进行拦截器方面的配置。struts.xml 文件的具体内容如代码 15.5 所示。

代码 15.5　实现 Action 配置：struts.xml

```
...
<struts>
    <constant    name="struts.custom.i18n.resources"    value="message">
    </constant>
    <constant name="struts.multipart.saveDir" value="c:\"></constant>
    <package name="struts2" extends="struts-default">
        <!--对类 UploadAction 进行配置-->
        <action name="upload" class="com.cjg.action.UploadAction">
            <result name="success">/uploadResult.jsp</result>
            <result name="input">/uploadfileup.jsp</result>
            <!--设置拦截器-->
            <interceptor-ref name="fileUpload">
                <!--设置允许上传文件的大小-->
                <param name="maximumSize">409600</param>
                <!--设置允许上传文件的类型-->
                <param name="allowedTypes">
                    application/vnd.ms-powerpoint
                </param>
            </interceptor-ref>
            <interceptor-ref name="defaultStack"></interceptor-ref>
        </action>
...
    </package>
</struts>
```

【代码解析】

- 在具体配置 Struts 2.x 内置上传文件拦截器时，可以从关于 Struts 2.x 框架核心包（struts2-core-2.jar）的 struts-default.xml 文件中，查询到名为 fileUpload 的拦截器。

```
       <interceptors>
...
          <interceptor name="fileUpload" class="org.apache.struts2.
          interceptor.FileUploadInterceptor"/>
...
       </interceptors>
```

注意：如果想深刻地理解 Struts 2.x 框架的上传文件功能，可以阅读 org.apache.struts2.interceptor.FileUploadInterceptor 类。

- 在配置关于 Struts 2.x 框架的一些信息时，可以查询 Struts 2.x 的核心包（struts2-core-2.jar）中名为 org.apache.struts2 包中的 default.properties 文件。属性 struts.i18n.encoding 用来设置编码方式，默认为 UTF-8；而 struts.multipart.saveDir 用来设置缓冲路径。

```
...
### This can be used to set your default locale and encoding scheme
# struts.locale=en_US
struts.i18n.encoding=UTF-8
...
# uses javax.servlet.context.tempdir by default
struts.multipart.saveDir=
...
```

注意： 当上传文件时，首先会把文件存放到缓冲路径中。如果不给赋值，则会出现“Unable to find ‘struts.multipart.saveDir’”错误。

如果想修改 default.properties 文件中的值，可以有两种方式。第一种方式：在 struts.xml 文件中修改，例如设置 struts.multipart.saveDir 的路径为 c:\。

```
<constant    name="struts.multipart.saveDir"
value="c:\"></constant>
```

第二种方式则是在如图 15.32 所示的目录中创建名为 struts.properties 的文件，例如如果想设置编码方式为中文：

```
struts.i18n.encoding = gbk
```

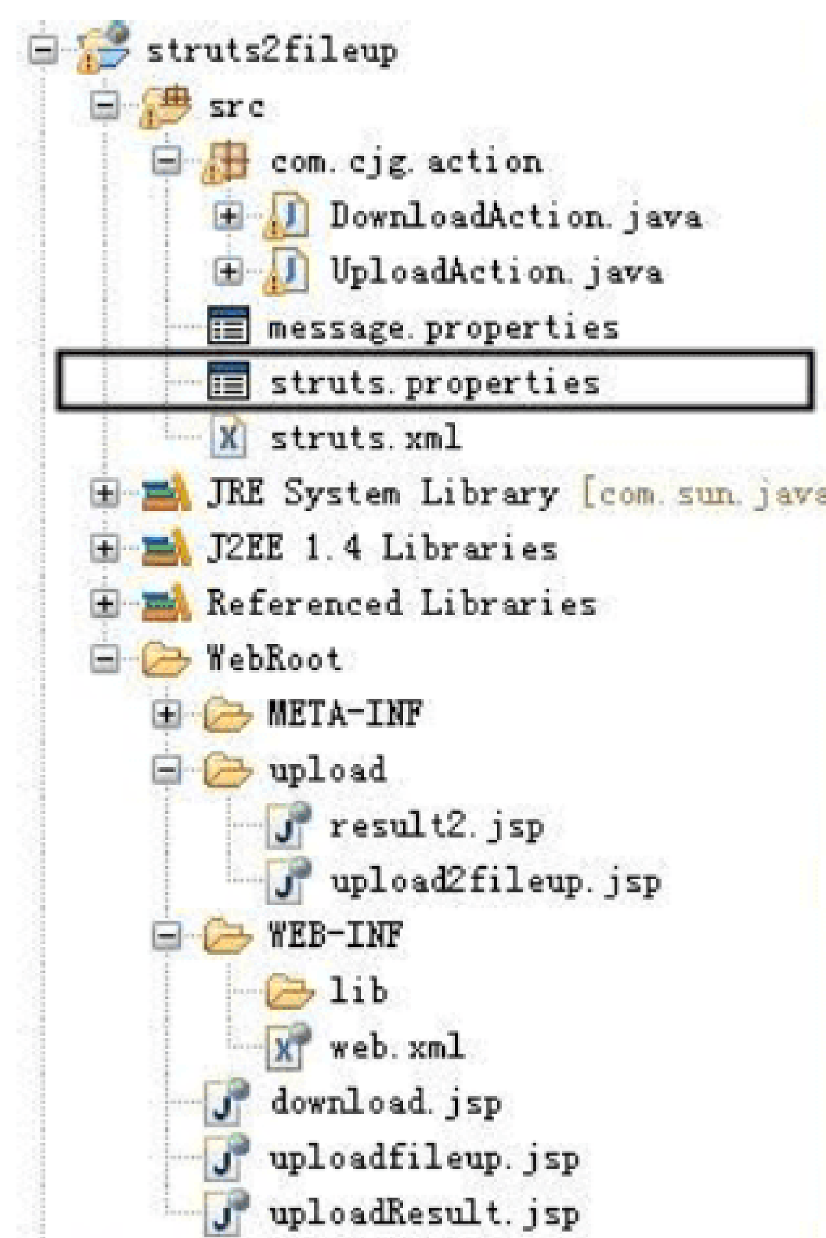


图 15.32　目录结构

15.5　多文件的上传

多文件上传有两种情况，第一种情况指定用户每次上传的文件数目；第二种情况不指定上传的文件数目，用户可以上传任意个文件，即可以上传一个文件也可以上传多个文件。显然第二种情况要比第一种情况灵活。

15.5.1　选择所要上传的文件

fileupload 页面用来让使用者实现选择所要上传的文件。只有具有一定权限的用户才能上传文件，所以在该页面中不仅要选择上传的文件，还必须填写用户名和密码。代码 15.6 用来获取所要上传的文件。

代码 15.6　选择所要上传的文件：upload.jsp

```
...
<!--添加和删除按钮方法-->
<script type="text/javascript">
function addMore()
{
    var td = document.getElementById("more");    <!--获取 ID 为 more 的元素-->

    var br = document.createElement("br");      <!--创建 br 元素-->
    var input = document.createElement("input"); <!--创建表单元素-->
    var button = document.createElement("input");<!--创建表单元素-->
    input.type = "file";                        <!--设置变量 input 的属性-->
    input.name = "file";
    button.type = "button";                     <!--设置变量 button 的属性-->
    button.value = "Remove";
    button.onclick = function()                 <!--单击变量 button 的方法-->
    {
```

```
        td.removeChild(br);                    <!--移除 br 元素-->
        td.removeChild(input);                 <!--移除 input 元素-->
        td.removeChild(button);                <!--移除 button 元素-->
    }
    td.appendChild(br);                        <!--添加 br 元素-->
    td.appendChild(input);                     <!--添加 button 的方法-->
    td.appendChild(button);                    <!--添加变量 button 的方法-->
}
</script>
</head>
<body>
...
        <s:form action="upload" theme="simple" enctype="multipart/form-
        data">
            <table align="center" width="50%" border="1">
                    <!--用户输入框-->
                    <td>
                        username
                    </td>
                    <td>
                        <s:textfield name="username"></s:textfield>
                    </td>
                    <!--密码框-->
                    <td>
                        password
                    </td>
                    <td>
                        <s:password name="password"></s:password>
                    </td>
                    <td>
                        file
                    </td>
...
            </table>
        </s:form>
...
```

【代码解析】

- 上述代码与上传单文件的 upload.jsp 相比，多创建了一个名为 addMore()的 JavaScript 函数。在 addMore()方法中，首先获取 ID 值为 more 的 td 元素，然后创建了实现回车换行功能的元素 br 和两个 input 类型的元素 input 与 button，接着配置元素 input 为 file 属性和元素 button 的属性为 button，最后通过 appendChild()方法把上述 3 个元素当作子元素添加到 td 元素中。
- 除了添加方法，还需要实现删除的方法。与添加方法基本相同，即通过 removeChild()方法删除元素 br、input 和 button。
- 上述添加和删除方法成功执行的前提条件，就是修改<td>元素的代码如下：

```
                    <td id="more">
                        <s:file name="file"></s:file><input type="button"
value="Add More.." onclick="addMore()">
                    </td>
```

由于需要定位到该<td>元素，所以需要为其设置 ID 属性的值。当单击 Add More 按钮就会触发名为 addMore()方法。

❑ 最后，上述代码的显示效果如图 15.33 所示。

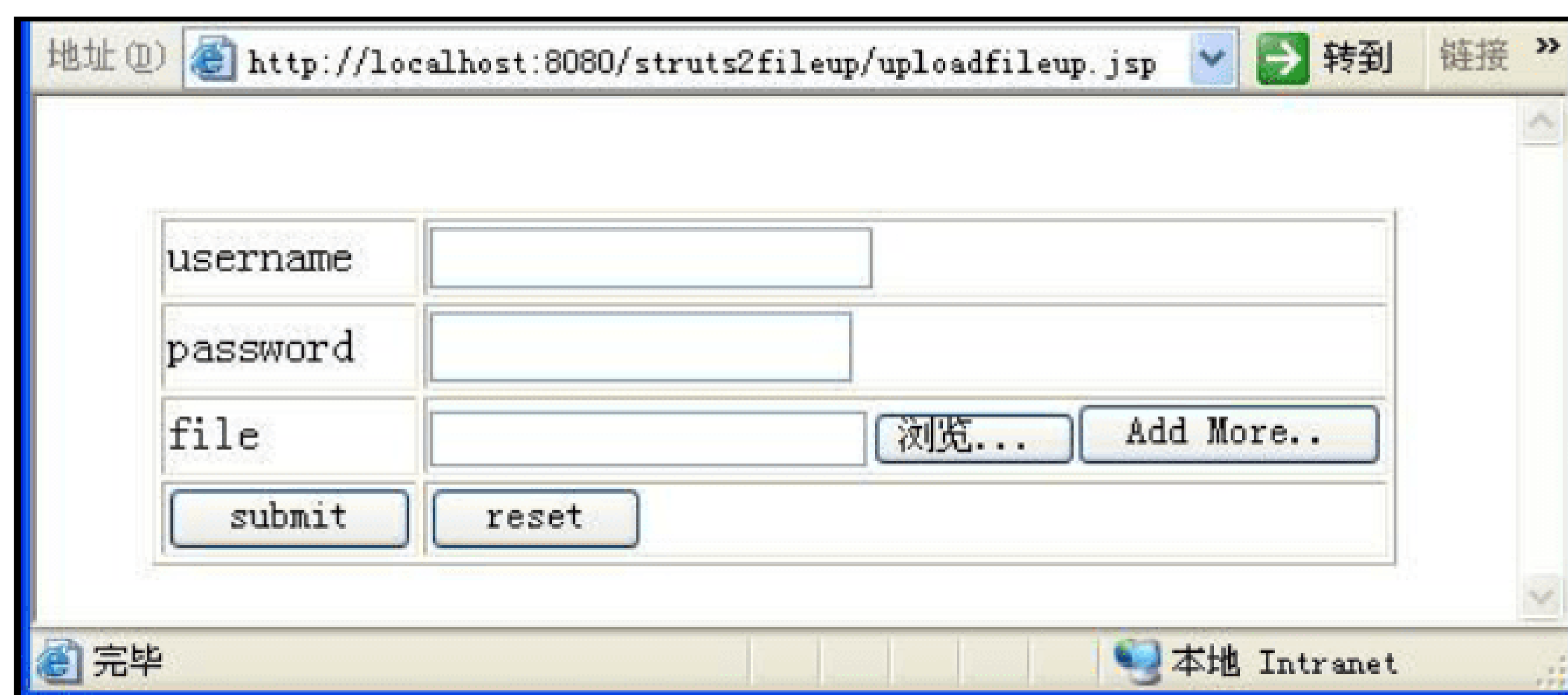


图 15.33　上传文件页面

15.5.2　实现文件的上传功能

实现文件的上传功能，首先通过输入流要获取 upload.jsp 页面传送过来的数据流信息，然后再通过输出流把文件信息存储到服务器的存储器中。代码 15.7 用来实现文件的上传功能。

代码 15.7　实现文件的上传：UploadAction.java

```
...
public class UploadAction extends ActionSupport
{
    private String username;                    //创建 username 属性
    private String password;                    //创建 password 属性
    private List<File> file;                    //创建 file 属性
    private List<String> fileFileName;          //创建 fileFileName 属性
    private List<String> fileContentType;       //创建 fileContentType 属性
    //省略上述属性的 get()和 set()方法
...
    public String execute() throws Exception    //编写 execute()方法
    {
        for (int i = 0; i < file.size(); ++i)
        {
            InputStream is = new FileInputStream(file.get(i));//创建输入流
            //创建父目录
            String root = ServletActionContext.getRequest().getRealPath(
                    "/upload");
            //创建输出流的目的文件
            File destFile = new File(root, this.getFileFileName(i));
            //创建输出流
            OutputStream os = new FileOutputStream(destFile);
            byte[] buffer = new byte[400];      //创建中间字节数组
            int length = 0;
            while ((length = is.read(buffer)) > 0)
                                                //实现输入流到输出流的转换
            {
                os.write(buffer, 0, length);
            }
            is.close();                             //关闭输入流
```

```
            os.close();                                 //关闭输出流
        }
        return SUCCESS;
    }
}
```

接着在 struts.xml 文件中对该 Action 程序进行配置。

```
<package name="struts2" extends="struts-default">
    <!--配置 Action-->
    <action name="upload" class="com.cjg.action.UploadAction">
        <result name="success">/uploadResult.jsp</result>
        <result name="input">/uploadfileup.jsp</result>
        <!--配置拦截器-->
        <interceptor-ref name="fileUpload">
            <param name="maximumSize">409600</param>
            <param name="allowedTypes">
                application/vnd.ms-powerpoint
            </param>
        </interceptor-ref>
        <interceptor-ref name="defaultStack"></interceptor-ref>
    </action>
```

【代码解析】

- 与上传单文件不同的地方就在于，属性 file、fileFileName 和 fileContentType 这 3 个属性的类型都变成 List 集合类型。这 3 个属性集合中的元素是一一对应的，即 file 集合中的第一个元素对应于 fileFileName 集合和 fileContentType 集合中的第一个元素。
- 在 execute()方法中经过 for 循环把集合中的每个元素都存储到目的地址。

为了配置上传文件的大小和类型，可以在拦截器 fileUpload 中通过参数 maximumSize 和 allowedTypes 来实现。如何知道这两个参数呢？可以通过查看 org.apache.struts2.interceptor.FileUploadInterceptor 类源代码知道其定义了如下两个属性。

```
...
Protected Long maximumSize;
Protected String allowedTypes;
...
```

如何设置允许上传类型（allowedTypes）的值呢？可以通过 Tomcat 的安装目录\conf\web.xml 文件中的如下内容获取。

```
...
   <mime-mapping>                                 <!--关于 avi 类型标记-->
       <extension>avi</extension>
       <mime-type>video/x-msvideo</mime-type>
   </mime-mapping>
...
   <mime-mapping>                                 <!--关于 bmp 类型标记-->
       <extension>bmp</extension>
       <mime-type>image/bmp</mime-type>
   </mime-mapping>
   <mime-mapping>                                 <!--关于 class 类型标记-->
       <extension>class</extension>
       <mime-type>application/java</mime-type>
   </mime-mapping>
```

```
    <mime-mapping>                                  <!--关于 doc 类型标记-->
        <extension>doc</extension>
        <mime-type>application/msword</mime-type>
    </mime-mapping>
    <mime-mapping>                                  <!--关于 exe 类型标记-->
        <extension>exe</extension>
        <mime-type>application/octet-stream</mime-type>
    </mime-mapping>
    <mime-mapping>                                  <!--关于 gif 类型标记-->
        <extension>gif</extension>
        <mime-type>image/gif</mime-type>
    </mime-mapping>
    <mime-mapping>                                  <!--关于 jpg 类型标记-->
        <extension>jpg</extension>
        <mime-type>image/jpeg</mime-type>
    </mime-mapping>
...
    <mime-mapping>                                  <!--关于 ppt 类型标记-->
        <extension>ppt</extension>
        <mime-type>application/powerpoint</mime-type>
</mime-mapping>
```

即如果允许上传的文件类型为 ppt，则参数值必须为 application/powerpoint。

15.5.3　实现文件下载

在 Struts 2.x 框架中实现文件的下载是一个比较简单的事情，即当用户单击下载链接，就会发出一个 Action 请求。在 struts.xml 中不仅配置了处理该请求的 Action 类，而且还为其设置了一些必要参数：例如所要下载文件的路径和文件名等。DownloadAction.java 为用来处理下载请求的 Action 类，具体内容如代码 15.8 所示。

代码 15.8　实现文件的下载：DownloadAction.java

```
...
public class DownloadAction extends ActionSupport {
    public InputStream getDownloadFile() {              //获取输入流
        return ServletActionContext.getServletContext().getResource-
        AsStream(
                "/upload/上传文件名字.ppt");
    }
    public String execute() throws Exception {          //编写 execute()方法
        return SUCCESS;
    }
}
```

接着在 struts.xml 文件中对该 Action 程序进行配置。

```
<action name="download" class="com.cjg.action.DownloadAction">
    <result name="success" type="stream">
        <param name="contentType">
            application/vnd.ms-powerpoint
        </param>
        <param name="contentDisposition">
            filename="上传文件名字.ppt"
        </param>
        <param name="inputName">downloadFile</param>
    </result>
</action>
```

【代码解析】

在具体配置 DownloadAction 类的拦截器参数时，除了需要设置必要属性外，元素<result>的类型 type 必须为 stream。

```
<result name="success" type="stream">
```

🔔**注意：**参数 inputName 的值必须为 getDownloadFile()方法名的后半部分，同时第一个字母必须为小写。

从关于 Struts 2.x 框架的核心包（struts2-core-2.jar）的 struts-default.xml 文件中，查询到名为 stream 的下载文件拦截器。

```
<result-type name="stream" class="org.apache.struts2.dispatcher.
StreamResult"/>
```

打开 org.apache.struts2.dispatcher.StreamResult 类的源代码，可以发现其中定义了 4 个变量。

```
...
Protected String contentType=”text/plain”;
Protected String contentLength”;
Protected String contentDisposition=”inline”;
Protected String inputName=”inputStream”;
Protected String inputStream inputStream;
Protected int bufferSize=1024;
...
```

其中属性 contentType 用来设置下载文件类型，属性 contentDisposition 用来设置下载文件的名字，属性 inputName 用来设置下载文件的输入流，属性 bufferSize 用来设置下载文件时的缓冲区大小。

实现文件下载页面的具体内容如代码 15.9 所示。

代码 15.9　下载文件页面：download.jsp

```
...
    <body>
        <s:a href="/struts2fileup/download.action">download</s:a>
    </body>
...
```

🔔**注意：**在上述代码中，单击 download 链接时，就会运行关于 download.action 请求的类。

15.6　小　　结

本章主要介绍文件的上传和下载模块，本系统基于 Struts 2.x+Components-FileUpload 共同构建而成。

为了让读者深入理解文件下载和上传模块，首先通过文件上传组件 Components-FileUpload 实现一个简单上传文件的例子，接着通过 Struts 2.x+Components-FileUpload 框架实现单文件上传和多文件上传，最后还实现了文件下载功能。

第 16 章　网上投票系统（Struts 2.x+JFreeChart）

一个功能强大的网站系统一般都会包含调查模块，而对于调查模块来说最普通的表现形式就是网上投票系统。利用网上投票系统可以在网络上完成对某个（些）问题的调查，然后根据投票系统的结果进行决策。本章将通过 Struts 2.x+JFreeChart 框架技术来介绍如何实现网上投票系统，该模块主要包含 3 种功能：实现投票、利用图像显示投票结果和保存投票信息。

16.1　网上投票系统原理

网上投票系统功能跟现实中的投票功能一样，不同的只是一个是实体而另一个是虚拟体。当网站管理员想了解一下浏览者的某些兴趣时，可以首先对这些兴趣做个投票系统，然后让浏览者参与投票，最后从投票的结果做出决策和决定。

16.1.1　网上投票系统框架分析

对于一个大型网站系统来说，实现一个可用的网上投票系统要考虑的情况十分复杂，例如：什么投票项目才吸引浏览者、如何要为投票项目做出充分的选项等。该章将会实现一个比较简单可用的网上投票系统，读者可以根据自己的需求自行进行完善。本系统结构框架如图 16.1 所示，项目目录如图 16.2 所示。

16.1.2　网上投票系统功能描述

本节将以直观的方式向读者介绍网上投票系统要实现的功能。这些功能包括实现投票、利用图形显示投票结果和保存每个浏览者的投票信息。为了能够说明该系统的效果列举了 3 个浏览者的投票结果。

1．第一个浏览者投票

浏览者首先通过浏览 selectvote.jsp 网页来实现投票，在该页面（如图 16.3 所示）上选择自己喜欢的运动后，然后单击“提交”按钮就可以用图形显示该浏览者所选的信息（如图 16.4 所示）。

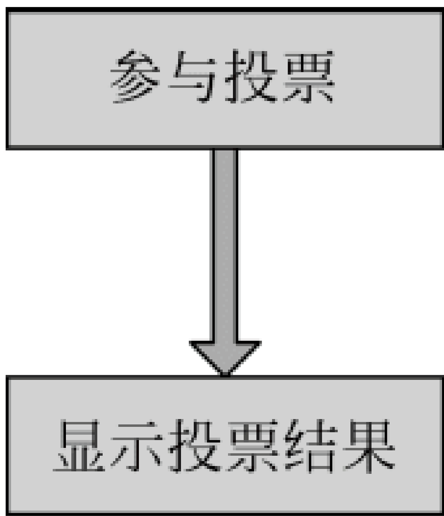


图 16.1　系统流程图

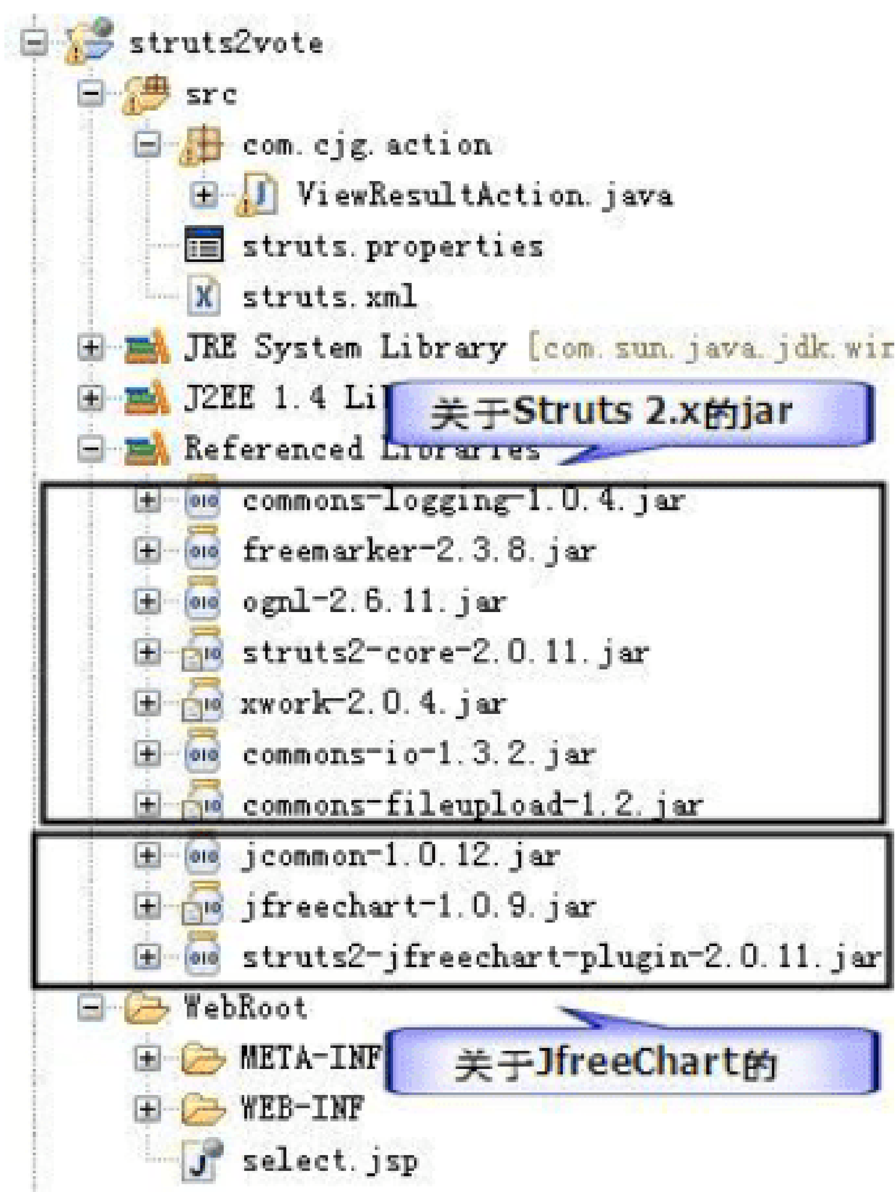


图 16.2　项目目录



图 16.3　第一个浏览者投票



图 16.4　显示第一次投票结果

2．第二个浏览者投票

当第二个浏览者通过浏览 selectvote.jsp 网页来实现投票，在该页面（如图 16.5 所示）上选择自己喜欢的运动后，然后单击“提交”按钮就可以用图形显示该浏览者与第一个浏览者所选的信息（如图 16.6 所示）。



图 16.5　第二个浏览者投票



图 16.6　显示第二次投票结果

3．第三个浏览者投票

当第三个浏览者通过浏览 selectvote.jsp 网页来实现投票时，在该页面（如图 16.7 所示）上选择自己喜欢的运动后，单击“提交”按钮就可以用图形方式显示该浏览者、第一个浏览者和第二个浏览者所选的信息（如图 16.8 所示）。

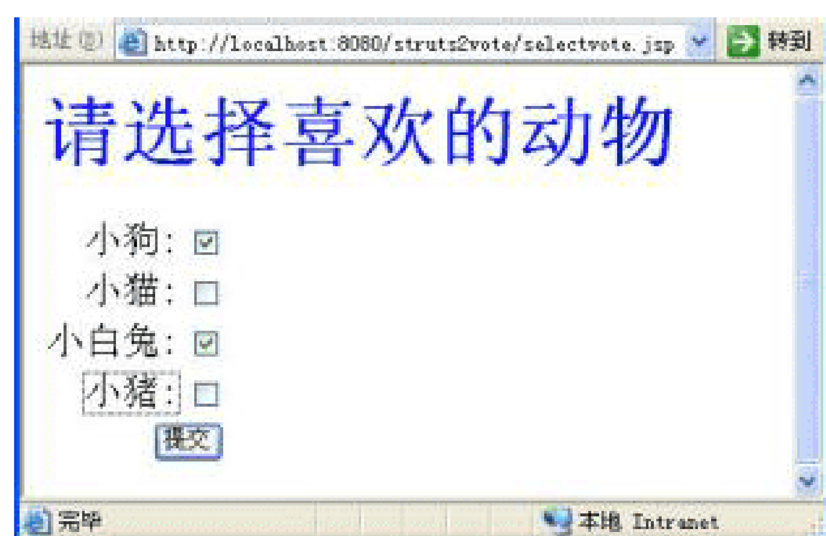


图 16.7　第三个浏览者投票



图 16.8　显示第三次投票结果

至此，就完成对网上投票系统所有功能的演示。

16.2　图表组件 JFreeChart

图表组件 JFreeChart 是 JFreeChart 公司在开源网站 SourceForge.net 上的一个项目，该组件用来提供 Java 方面（Application/Applet/Servlet/JSP）图形的解决方案。图表组件 JFreeChart 是一款功能强大的图形生成工具，可以直接生成后缀名为 PNG、JPG 等的图形文件。

16.2.1　下载图表组件（JFreeChart）和相关组件

虽然图表组件 JFreeChart 是一款免费的图形生成工具，但是其的 document 文档却是需要 40 美金才能获取。该组件的最新版本为 1.0.12，具体下载步骤如下。

（1）首先访问下载 JFreeChart 的官方网站（http://www.jfree.org/），如图 16.9 所示。



图 16.9　JFreeChart 组件首页

（2）打开 JFreeChart 的官方网站首页，选择 JFREECHART 导航栏后，就可以打开关于 JFreeChart 的页面，如图 16.10 所示。如果想下载该产品，直接在该页面单击 DOWNLOAD 导航栏就可以转到关于下载的页面。

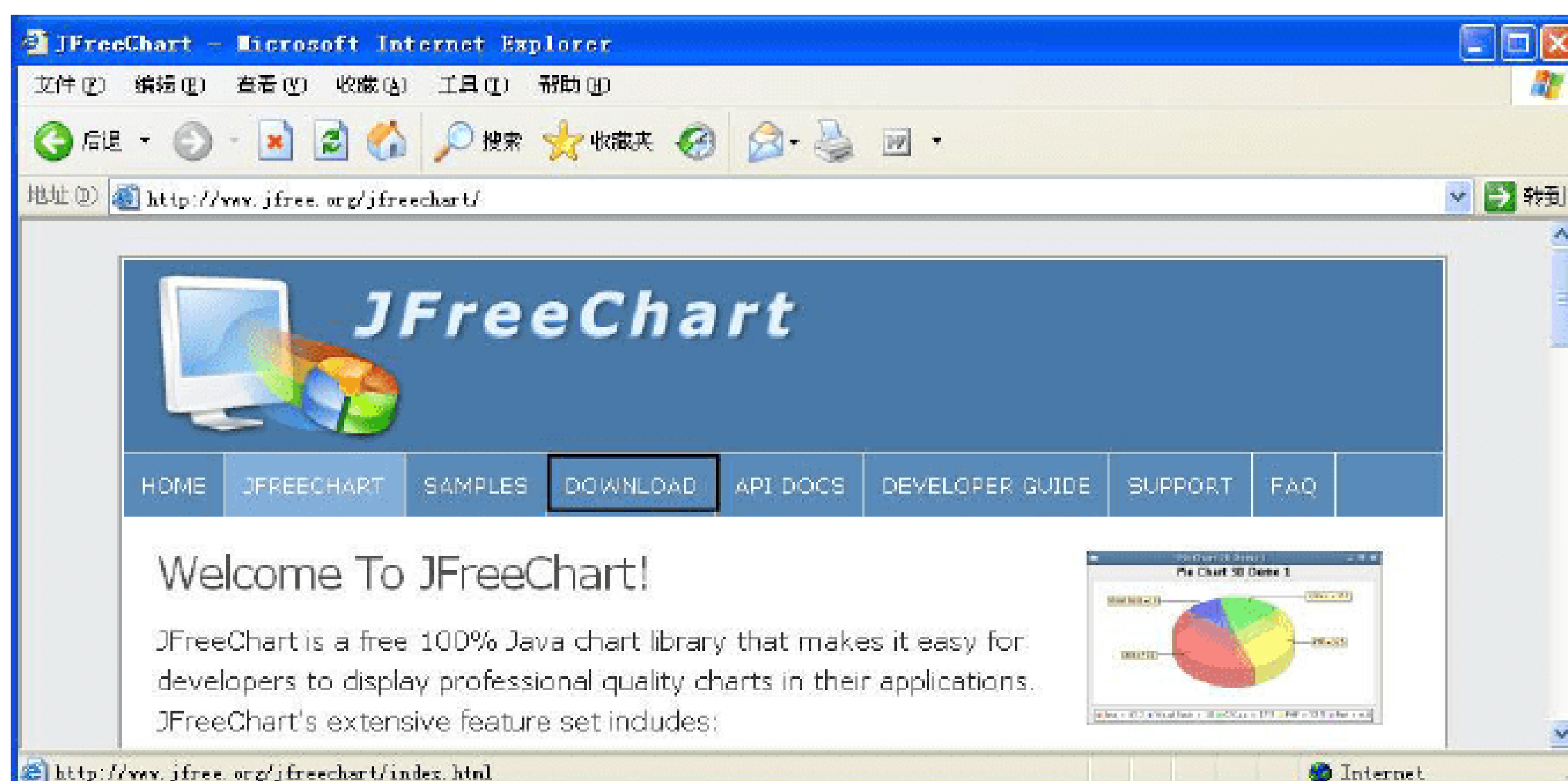


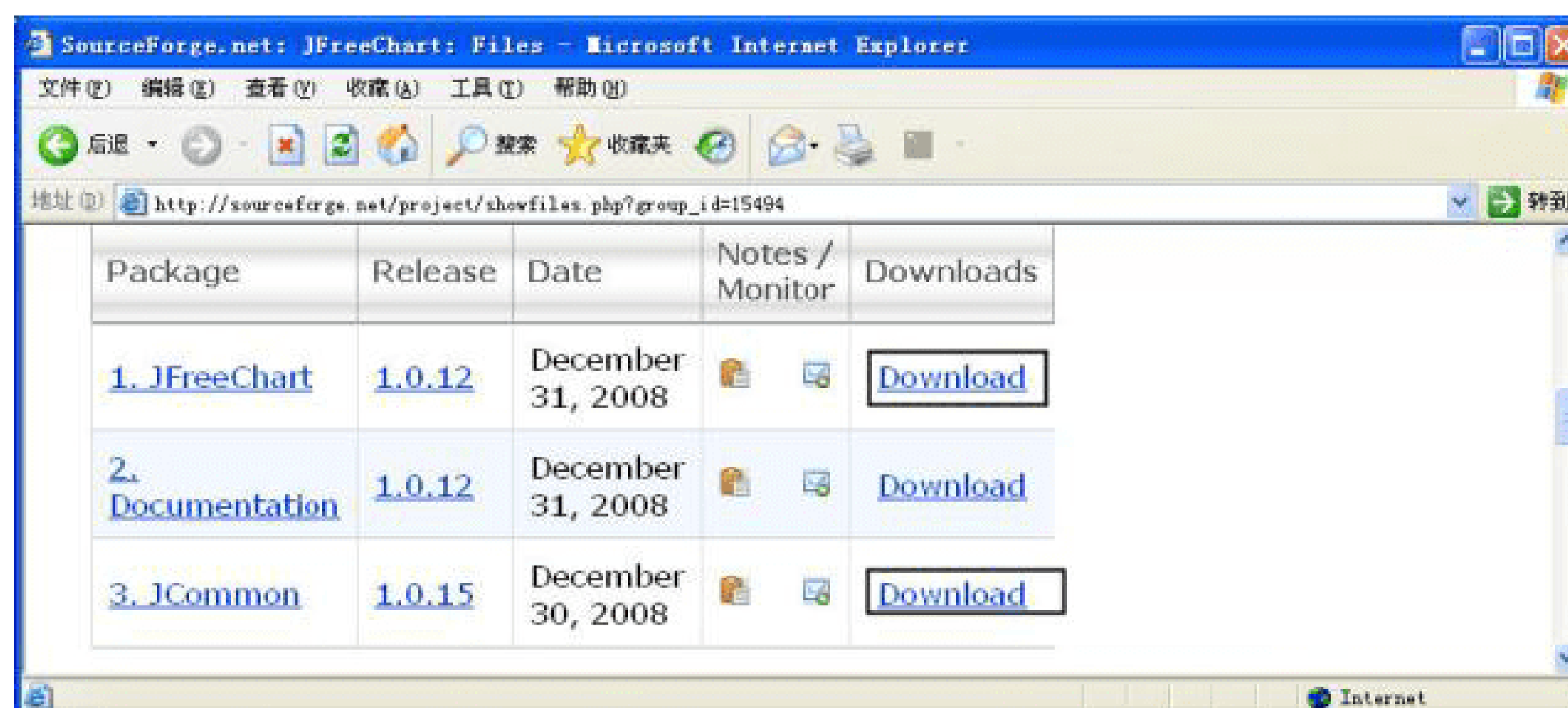
图 16.10　关于下载 Displaytag 标记库的页面

（3）在关于下载 JFreeChart 的页面（如图 16.11 所示）中单击 SourceForge file server 链接，就可以转到下载 JFreeChart 类型的页面。



图 16.11　下载页面

（4）在选择下载 JFreeChart 类型的页面中（如图 16.12 所示），分别单击 Download 链接下载 JFreeChar 和 JCommon 两个类型产品。虽然在 JFreeChart 组件的文件中已经包含了 Jcommon 组件的 jar 包，但是在具体开发中需要查看关于 JCommon 组件的帮助文档，因此还需要下载 JCommon 组件。



Package	Release	Date	Notes / Monitor	Downloads
1. JFreeChart	1.0.12	December 31, 2008		Download
2. Documentation	1.0.12	December 31, 2008		Download
3. JCommon	1.0.15	December 30, 2008		Download

图 16.12　下载 Displaytag 标记库

（5）当单击 Download 链接后，就会转到相应的关于软件类型的页面。在图 16.13 中单击 jfreechart-1.0.12.zip 链接下载关于 JFreeChart 的 jar 包，在图 16.14 中单击

jcommon-1.0.15.zip 链接下载关于 JCommon 的 jar 包。

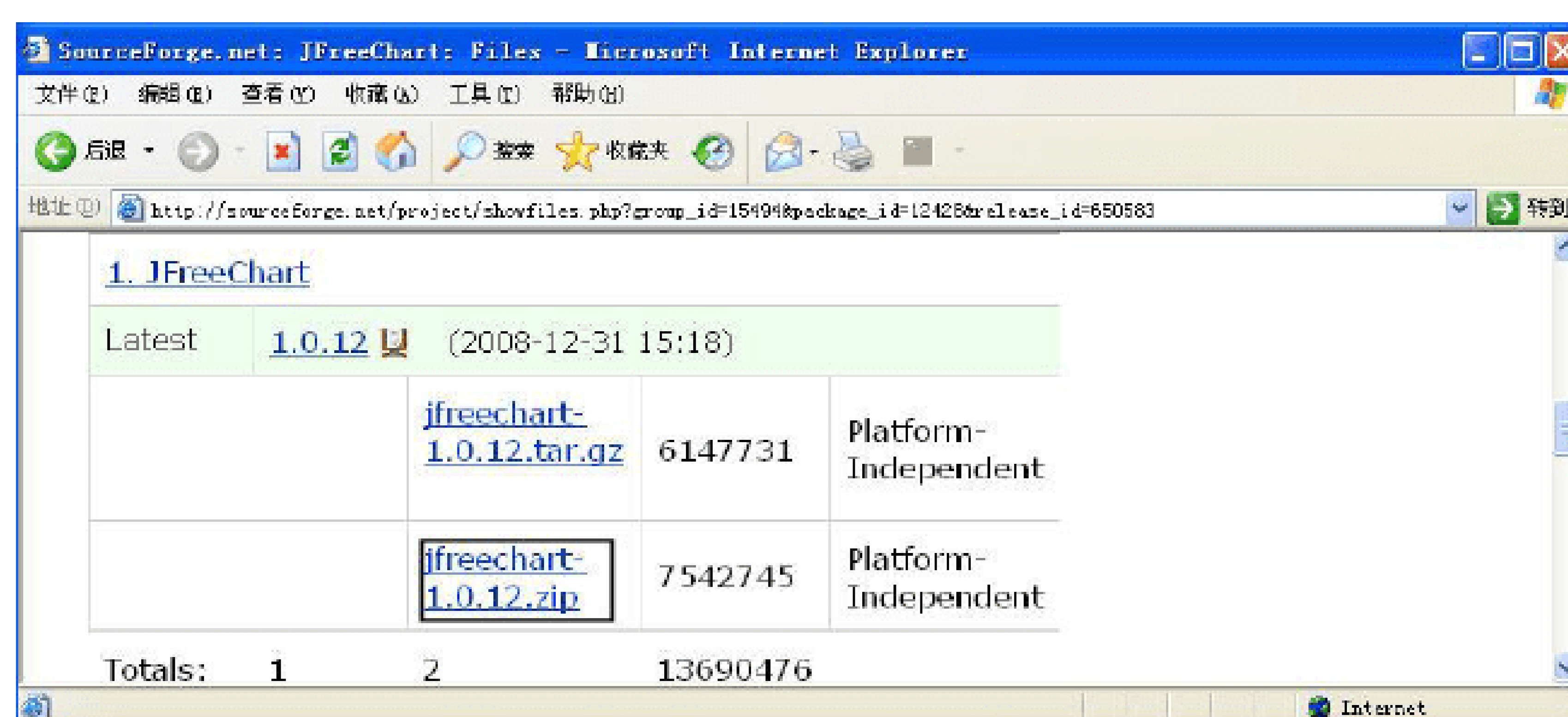


图 16.13　下载 JFreeChart 组件

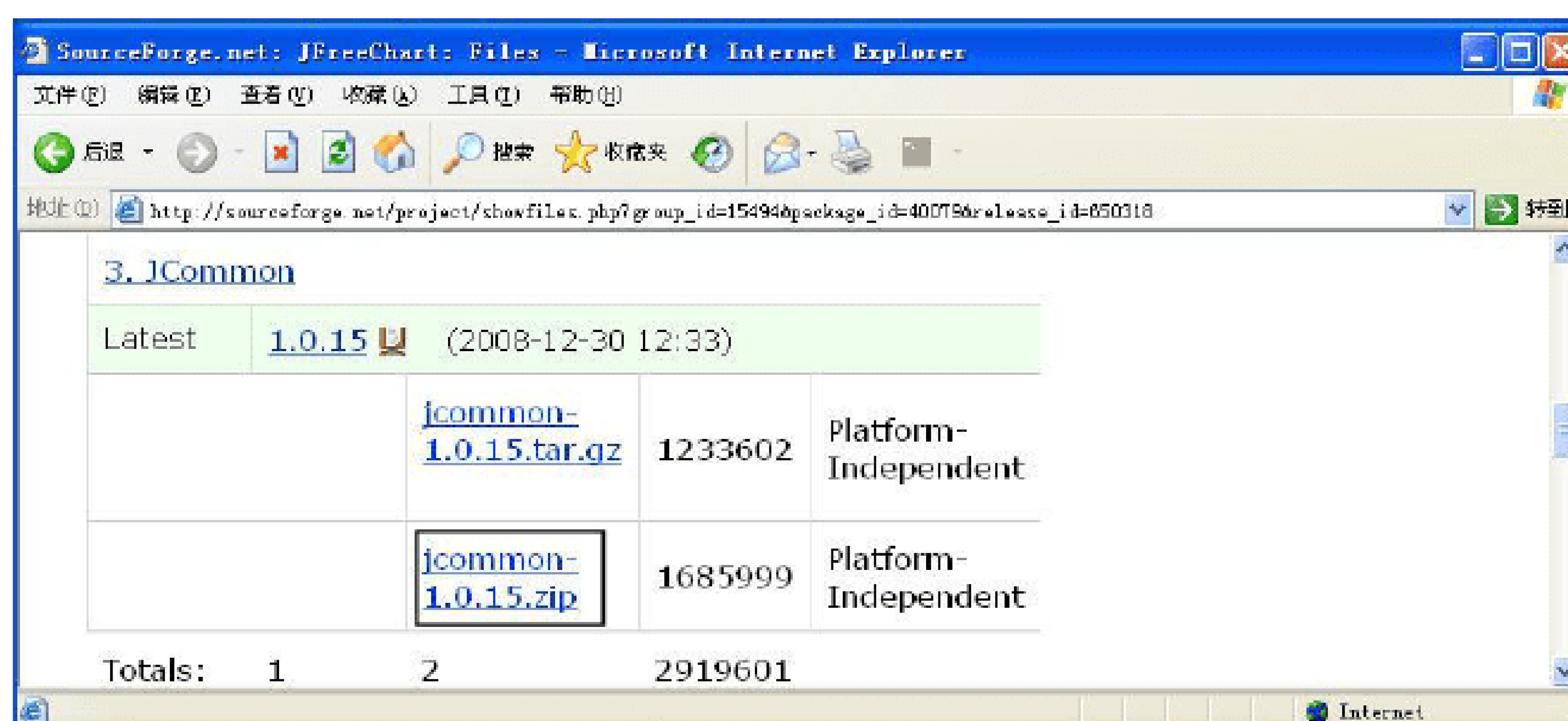


图 16.14　下载 JCommon 组件

至此，就完成对图表组件 JFreeChart 和该组件相关组件的下载。如果想在关于 Struts2.x 项目中使用图表组件 JFreeChar，还必须使用下载关于该组件的插件 JFreeChartPlugin 组件，具体步骤如下。

（1）首先访问下载 apache 组织的官方网站（http://www.apache.org/），如图 16.15 所示。在该页面中选择 Foundation 导航栏就可以进入 apache 组织的产品列表。

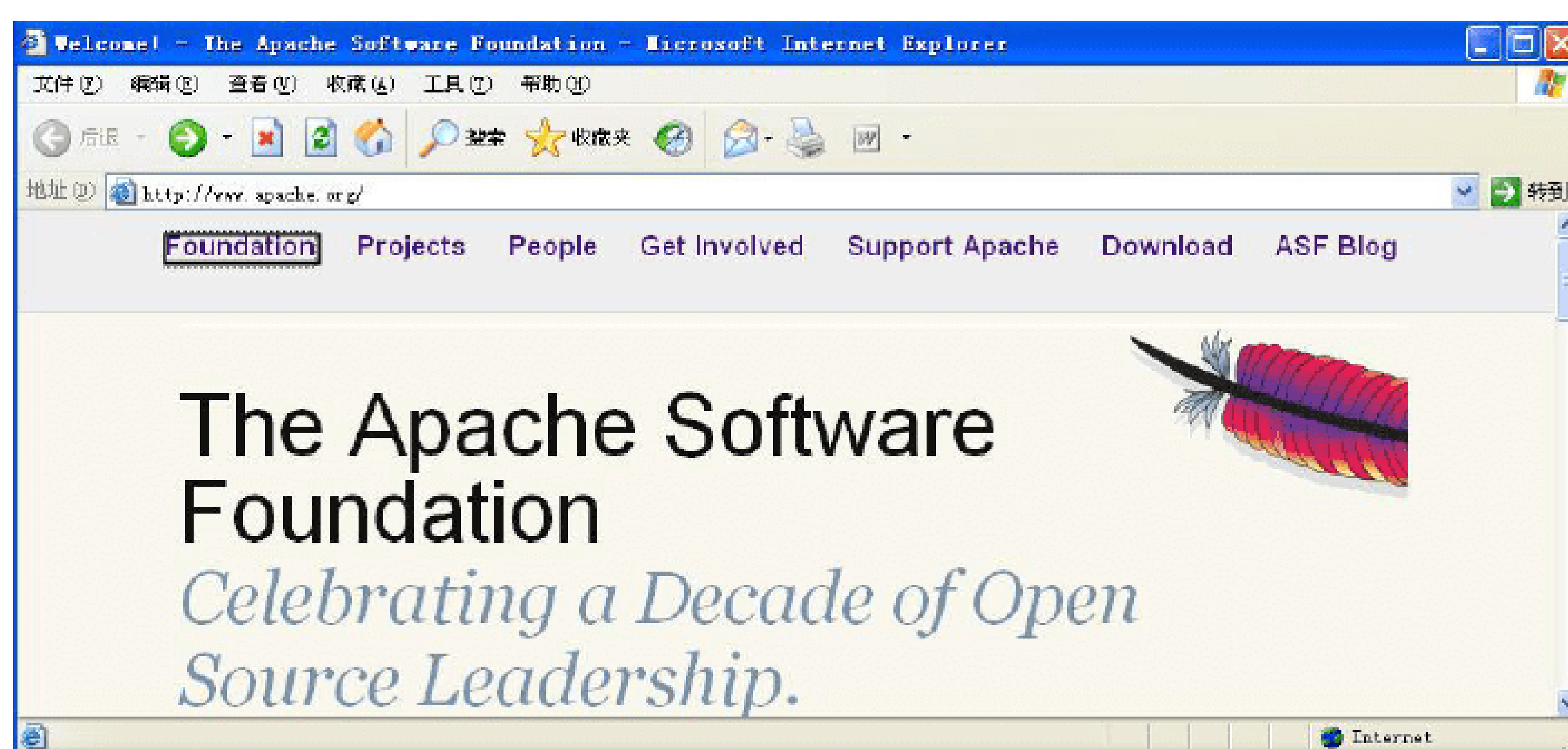


图 16.15　apache 组织的官方首页

（2）打开进入 apache 组织的产品列表页面（如图 16.16 所示）后，选择右边 Apache Projects 栏目中的 Struts 选项，就可以打开关于 Struts 的页面，如图 16.17 所示。在该页面中选择 Struts 2 导航栏就可以直接进入关于 Struts 2 的页面。

（3）在关于 Struts 2 的页面（如图 16.18 所示）中单击 Plugin Registry 图标，就可以转到关于 Struts 2 插件的页面。

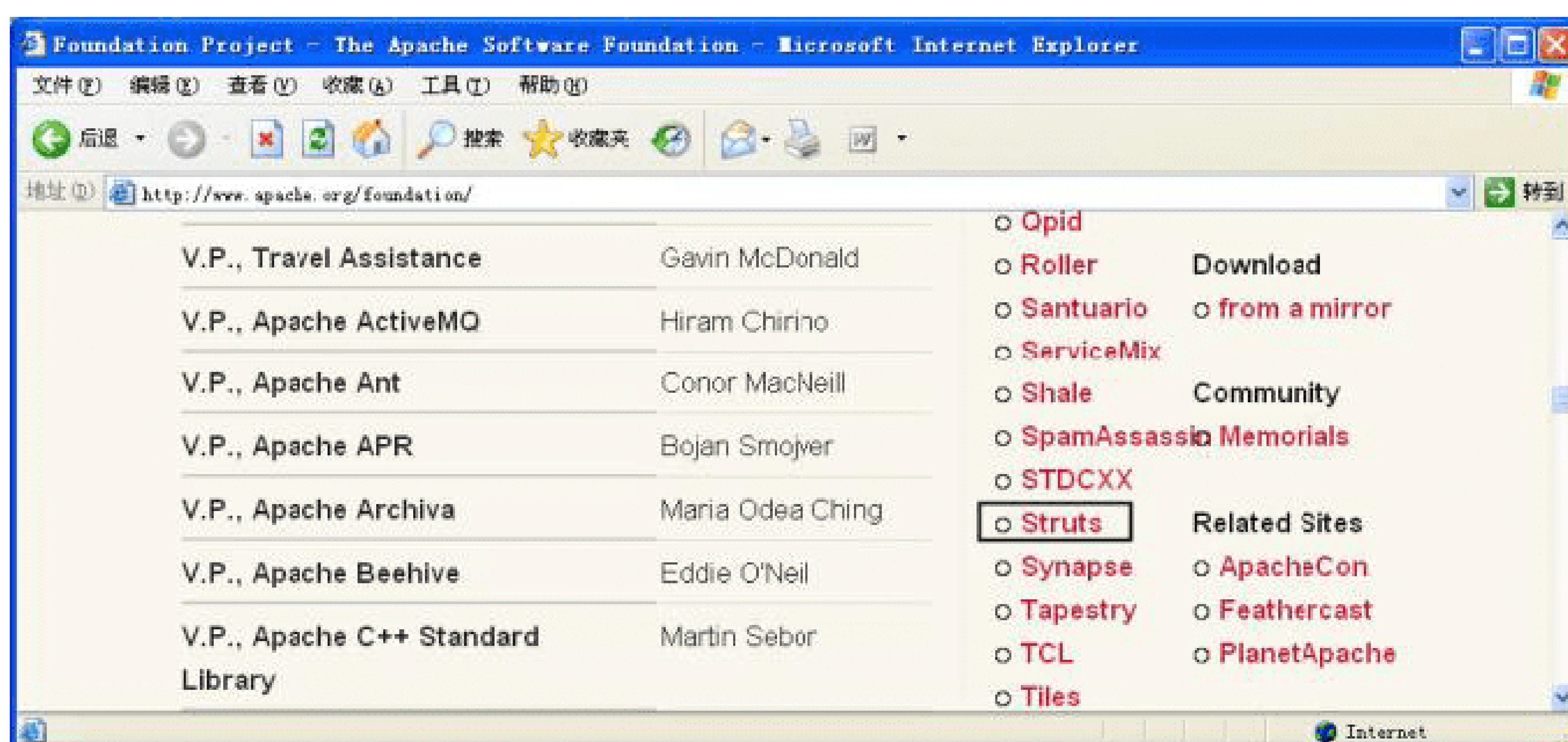


图 16.16　apache 组织产品列表的页面

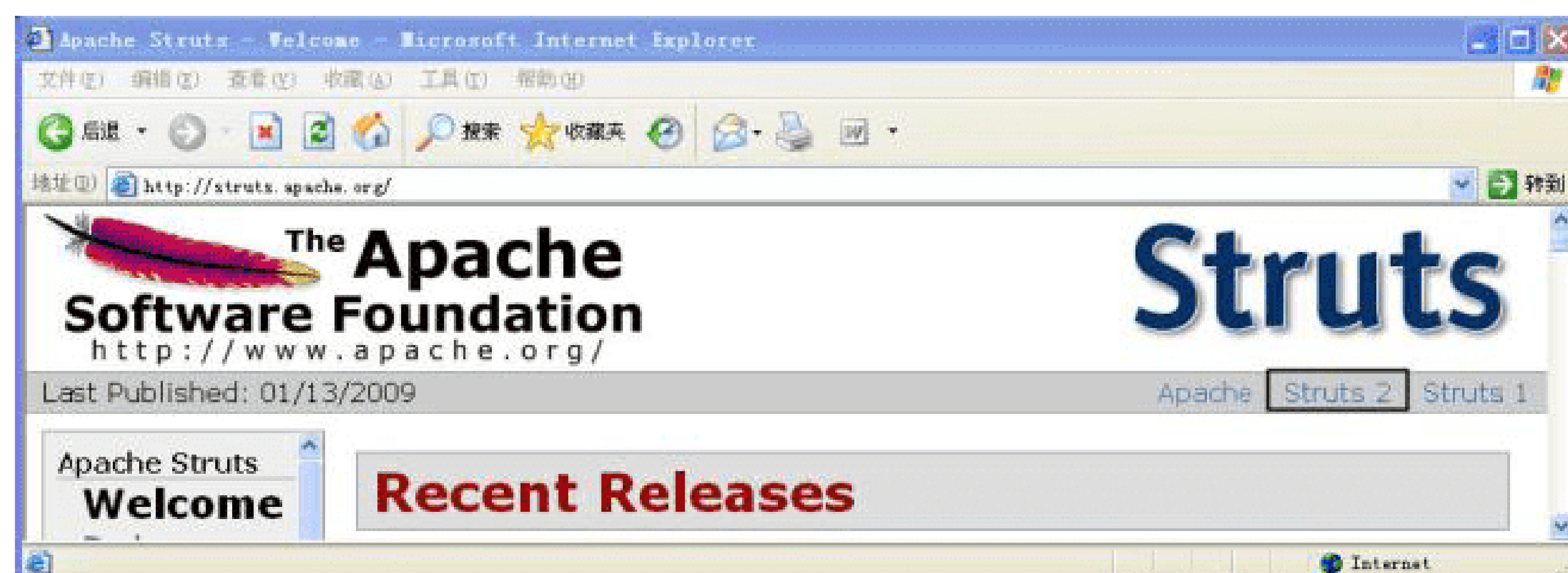




图 16.17　关于 Struts 的页面　　　　　图 16.18　关于 Struts 2 的页面

（4）在关于 Struts 2 插件的页面（如图 16.19 所示）中单击 JFreeChartPlugin 链接，就可以转到关于 JFreeChart 插件的页面，如图 16.20 所示，在该页面中单击 Download 后面的链接就可以下载 JFreeChartPlugin 组件。

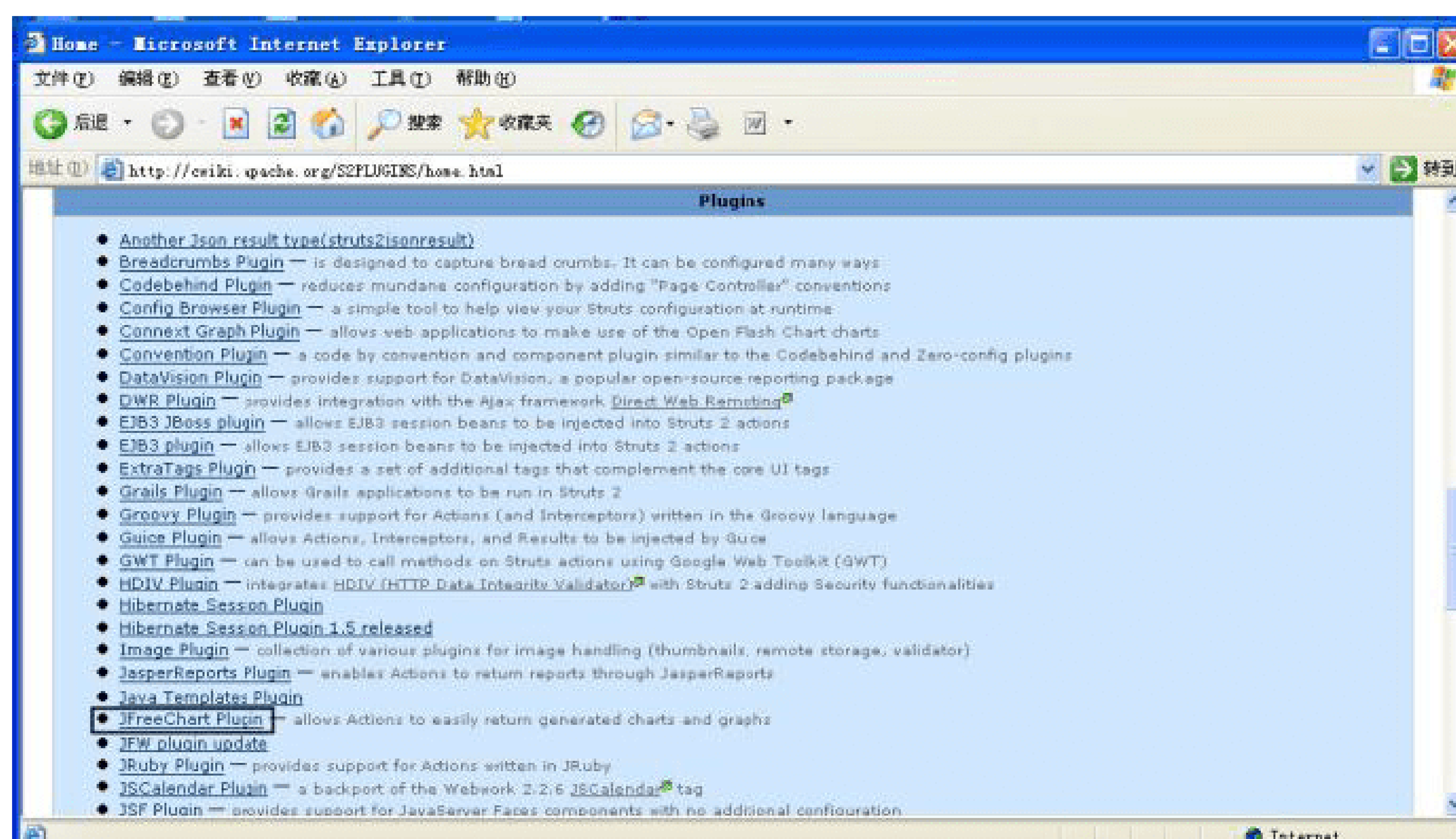


图 16.19　关于 Struts 2 插件的页面

至此，就完成对图表组件 JFreeChart 相关插件 JFreeChartPlugin 的下载。

16.2.2　了解和管理图表组件（JFreeChart）文档

16.2.1 节介绍了如何下载图表组件 JfreeChart 及其相关组件，下载完这些组件后就可以在 Java Web 项目中使用该框架。在具体使用之前，先解压 jfreechart-1.0.12.zip 文件，该压缩包目录如图 16.21 所示。各个文件的作用如下。

- tests：关于 JFreeChart 的单元测试代码。

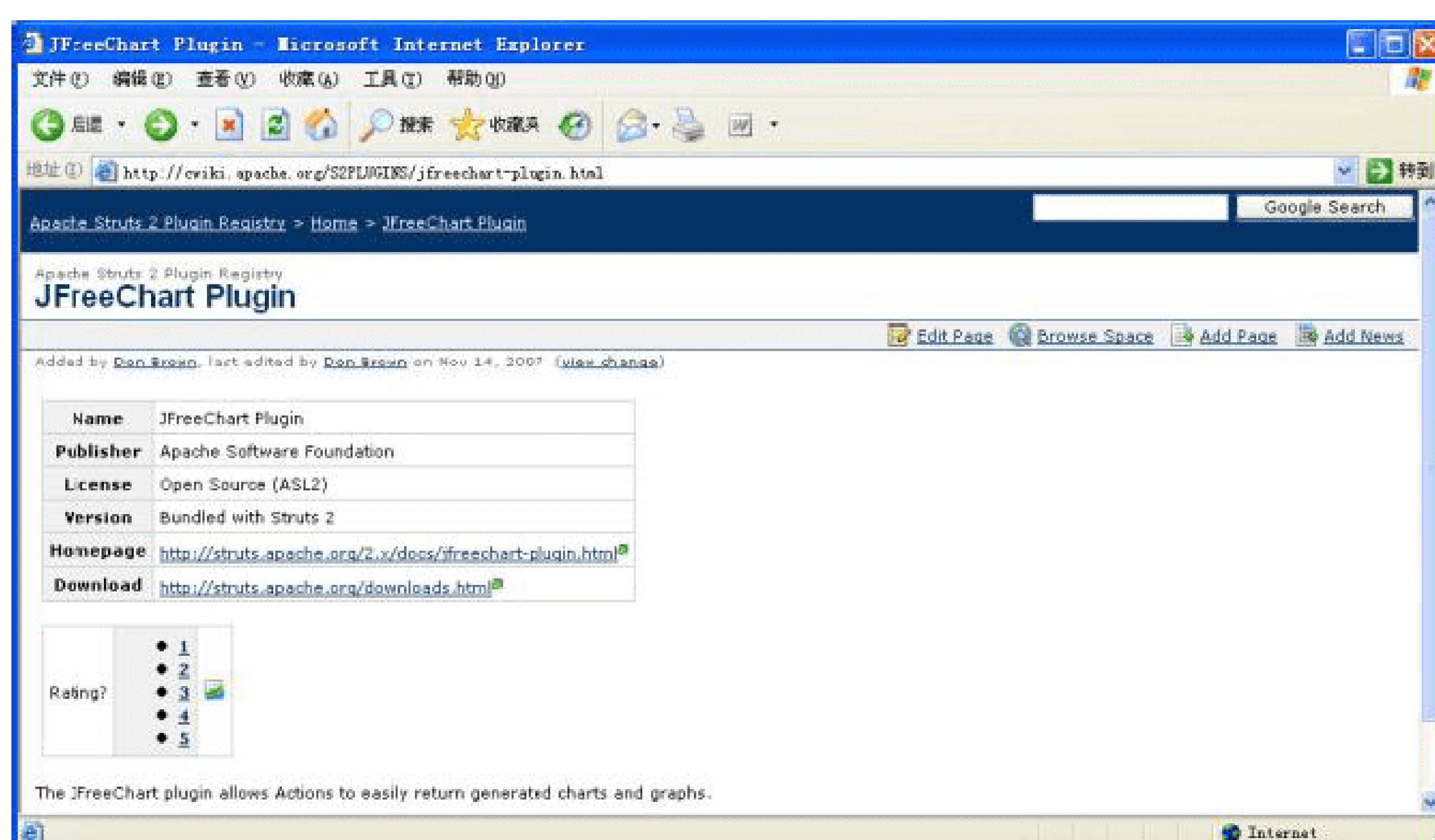


图 16.20　关于 JFreeChart 插件的页面

名称	大小	压缩后大小
..		
tests		
swt		
source		
lib		
experimental		
docfiles		
checkstyle		
ant		
jfreechart-1.0.12-demo.jar	771,230	713,393
README.txt	35,080	11,903
NEWS	26,874	9,450
maven-jfreechart-project.xml	2,766	1,012
licence-LGPL.txt	26,430	9,358
ChangeLog	369,168	54,415

图 16.21　目录结构

- ❑ source：关于 JFreeChart 的源代码。
- ❑ lib：关于 JFreeChart 的一些架包。
- ❑ docfiles：关于 JFreeChart 的相关文档。
- ❑ checkstyle：关于 JFreeChart 的 CSS 文件。
- ❑ ant：关于 JFreeChart 的 ant 脚本。
- ❑ jfreechart-1.0.12-demo.jar：双击可以执行的关于 JFreeChart 的演示实例。其界面如图 16.22 所示，在左边的项目中选择 JfreeChart|Area Charts|AreaChartDemol.java 就可以显示出相应的图形，如图 16.23 所示。

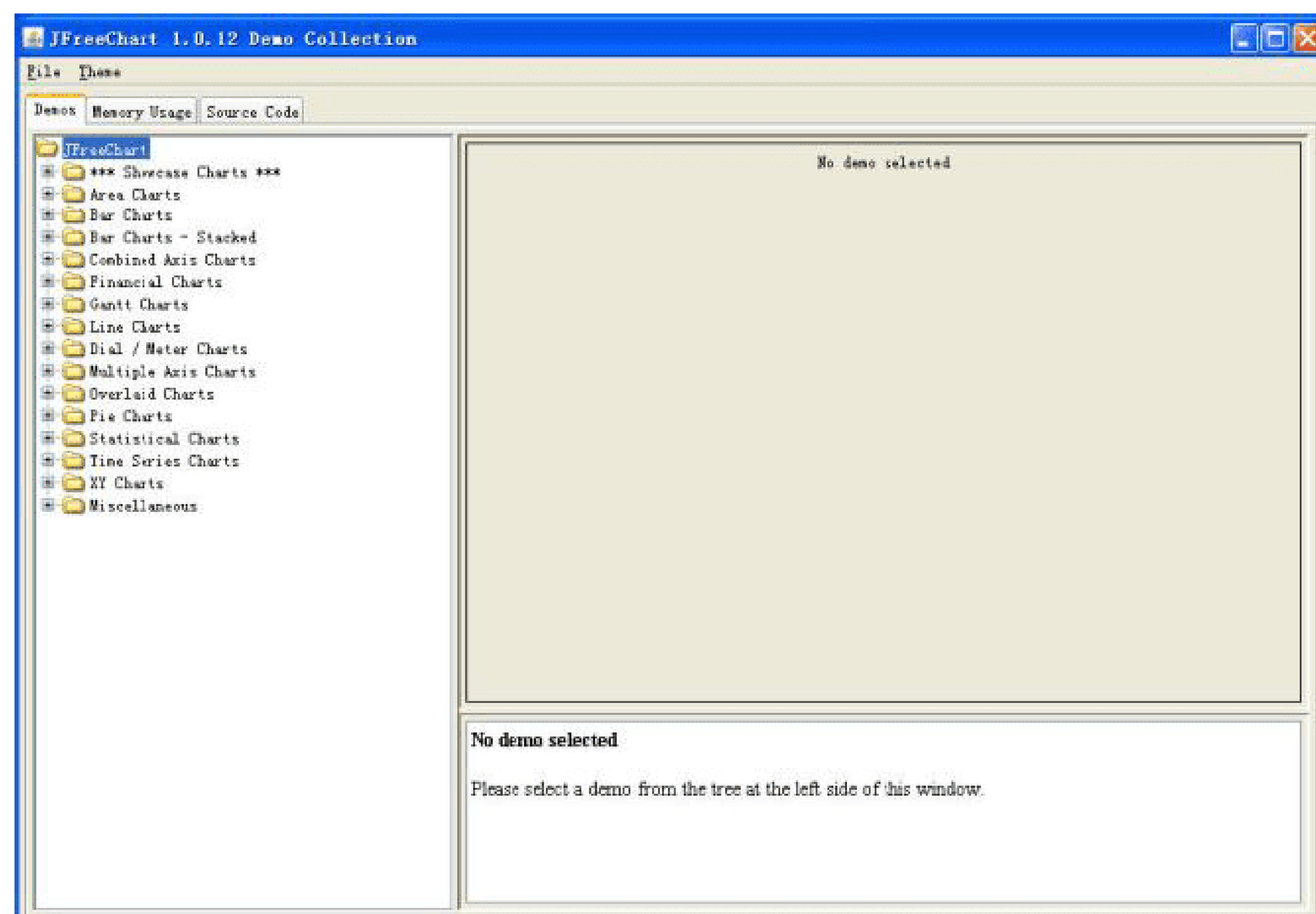


图 16.22　运行界面

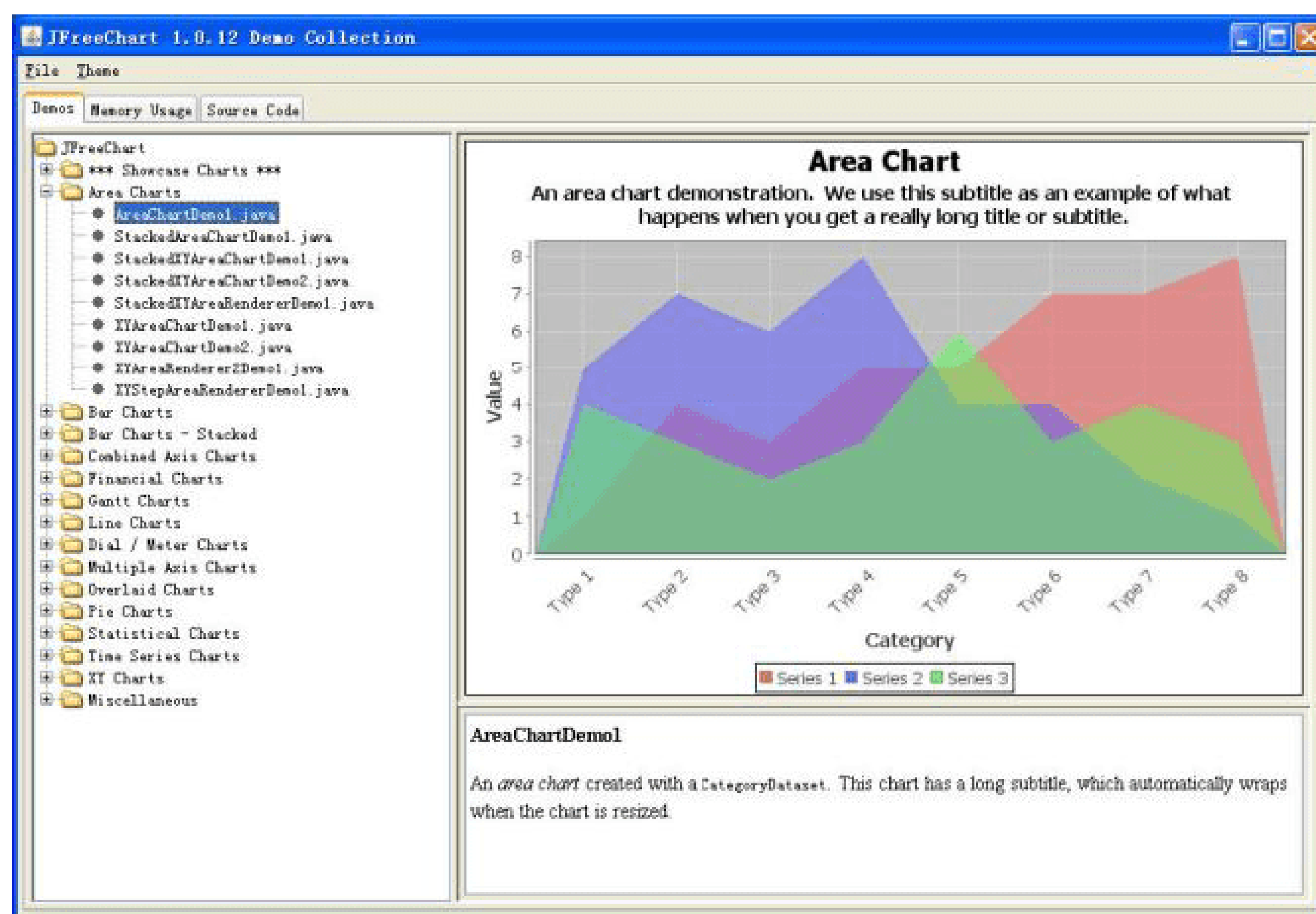


图 16.23　演示实例

如何通过 ant 文件的 XML 文件来生成关于 JFreeChart 的 HTML 语言的帮助文档？首先要安装 ant 软件，然后打开 Windows 的 Dos 窗口，在该窗口中通过命令进入 jfreechart-1.0.12\jfreechart-1.0.12\ant 文件夹中，然后通过命名 ant javadoc 把生成的帮助文档存放在 javadoc 文件夹中，如图 16.24 所示。具体的执行过程如图 16.25 所示。

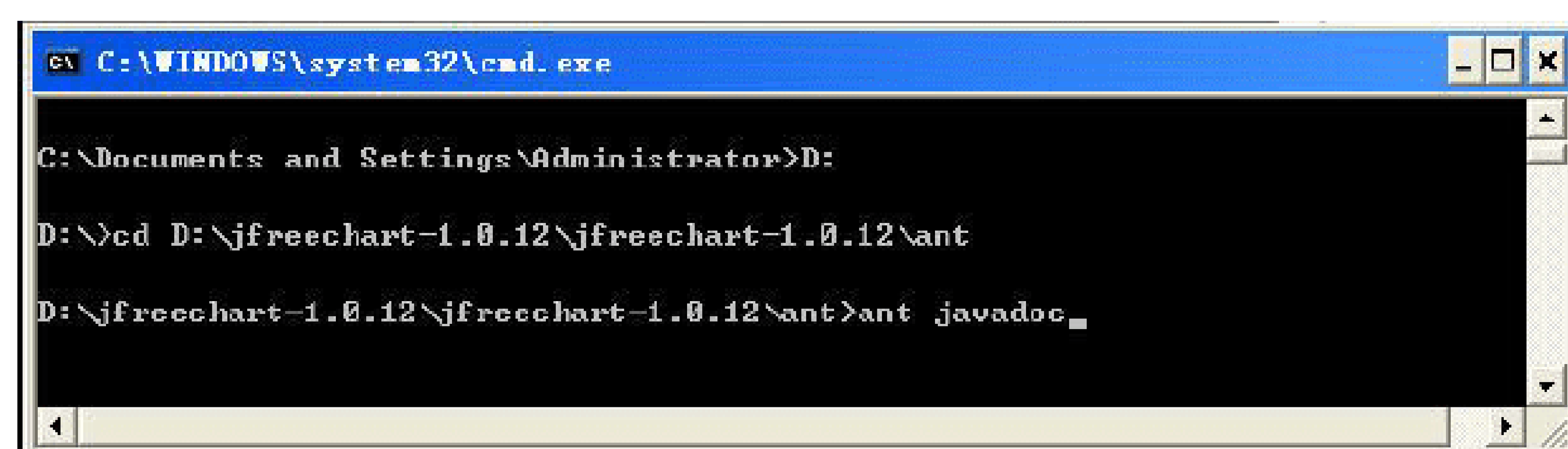


图 16.24　生成帮助文档命令

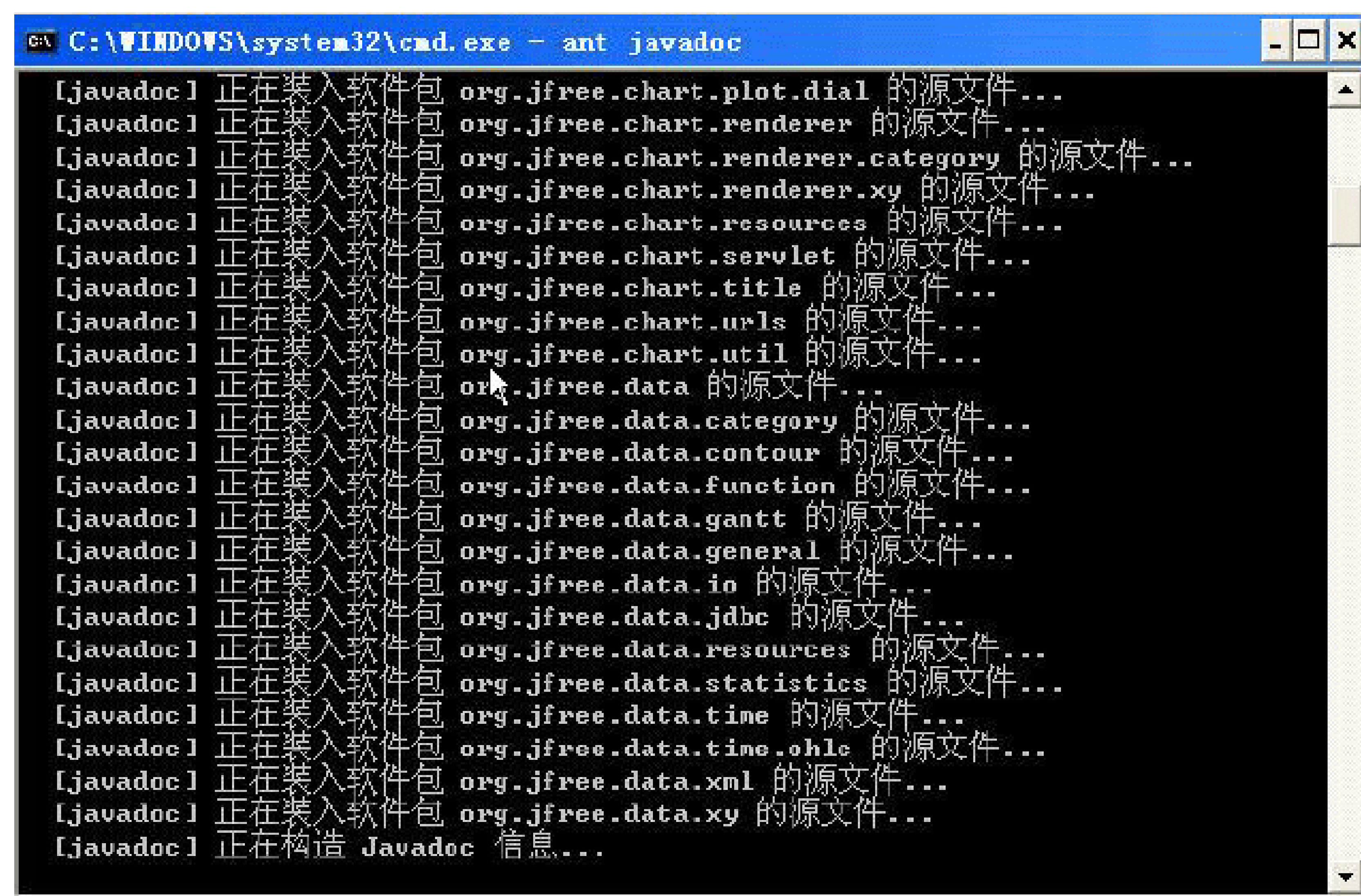


图 16.25　执行过程

为了使用方便，可以复制 jfreechart-1.0.12\jfreechart-1.0.12\lib 文件下的 jfreechart-1.0.12.jar 和 jcommon-1.0.15.jar 这两个 jar 包，到相应的文件夹中，然后把这两个 jar 包在 MyEclipse 开发环境中专门建立一个名为 JFreeChart 用户库，如图 16.26 所示。

至此，就完成了关于图表组件 JFreeChart 用户库的配置。

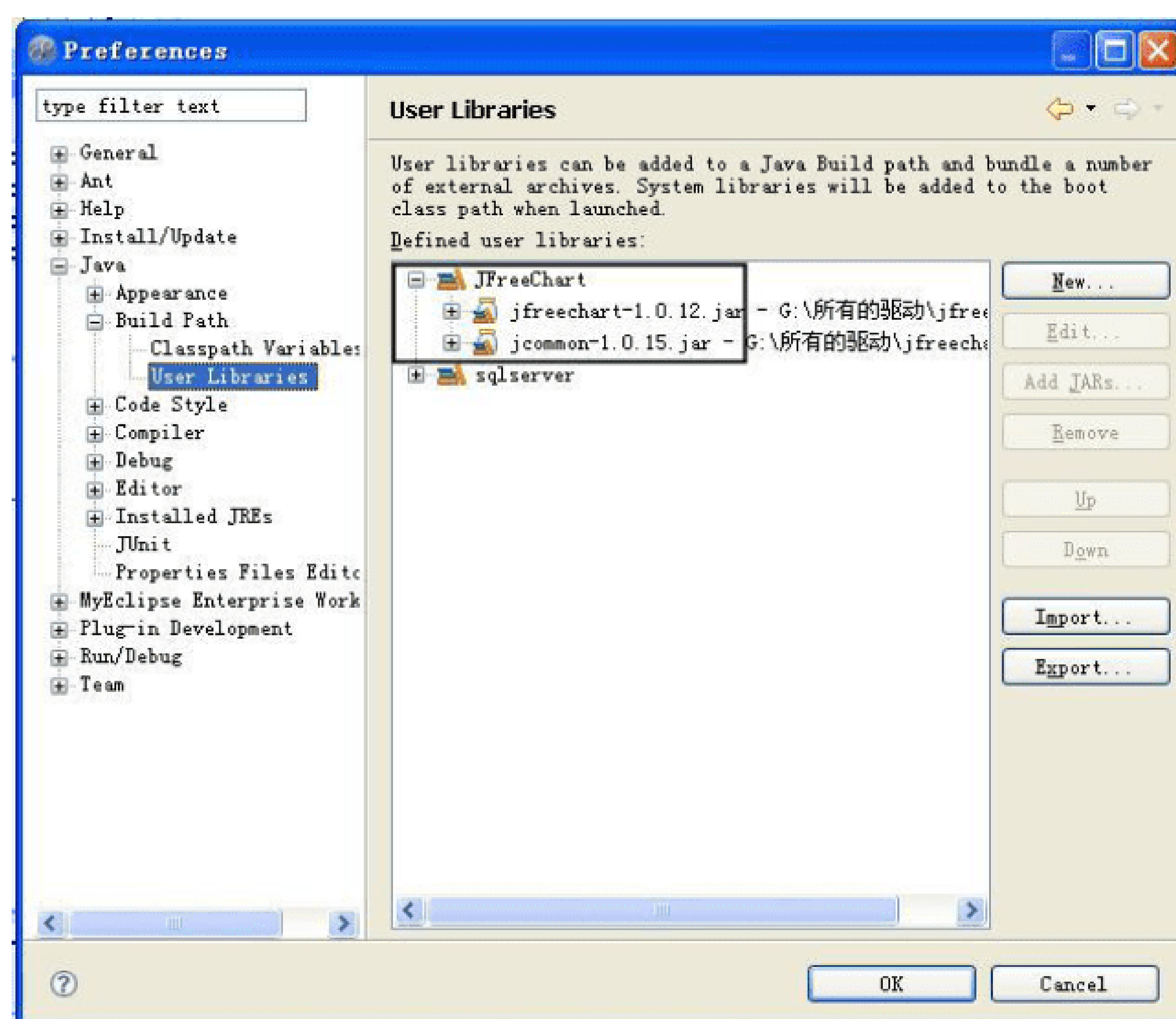


图 16.26　配置用户库对话框

16.3　初步使用图表组件（JFreeChart）

本节通过两个具体的事例来讲解如何使用 JfreeChart 组件：jfreechart 项目和 jfreecharweb 项目。jfreechart 项目主要用来讲解如何在 Java 应用程序中实现饼形图形，jfreecharweb 项目主要介绍如何在 Java Web 项目中实现饼形图形。

16.3.1　Java 应用项目实现饼形图形

JfreeChartTest.java 文件通过利用 JfreeChart 组件的相关类实现了饼形图形。如果想要让所要求的图形正确显示出来，一般要经历如下步骤。

（1）准备相应图形的数据集。

（2）创建出相应图形的 JfreeChart 对象。

（3）在 ChartFrame 对象上显示 JfreeChart 对象。

实现饼形图形文件的具体内容如代码 16.1 所示。

代码 16.1　实现饼形图形：JfreeChartTest.java

```
//导入相应包
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;

public class JFreeChartTest
{
    public static void main(String[] args)
    {
        DefaultPieDataset dpd = new DefaultPieDataset();
```

```
                                                //创建关于饼图的数据集对象
        dpd.setValue("1-25", 25);               //为数据集对象添加数据
        dpd.setValue("25-50", 25);              //为数据集对象添加数据
        dpd.setValue("50-75", 45);              //为数据集对象添加数据
        dpd.setValue("75-100", 10);             //为数据集对象添加数据
        //由工厂类创建一个 JfreeChart 对象
        JFreeChart chart = ChartFactory.createPieChart3D("OLD", dpd, true,
                true, false);
        //创建 ChartFrame 对象
        ChartFrame chartFrame = new ChartFrame("OLD", chart);
        chartFrame.pack();
        chartFrame.setVisible(true);
    }
}
```

【代码解析】

首先为所要画的饼形图创建 DefaultPieDataset()类型数据集 dpd，不同类型的图形对应于不同类型的数据集。接着通过 setValue()方法为数据集 dpd 添加数据，该方法定义如下：

```
setValue(java.lang.Comparable key, double value)
```

第一个参数一般为一个字符串，因为字符串类继承了 Comparable 类。

接着创建关于饼形图相应的 JfreeChart 对象。查看帮助文档可以发现在包 org.jfree.chart 中存在一个类 ChartFactory，该类存在许多方法，例如 createPieChart()用来创建关于 2D 饼形的 JFreeChart 对象，createPieChart3D()用来创建关于 3D 饼形的 JFreeChart 对象。其中 createPieChart()的定义如下：

```
public static JFreeChart createPieChart(java.lang.String title,
                                 PieDataset dataset,
                                 boolean legend,
                                 boolean tooltips,
                                 boolean urls
                                 )
```

第 1 个参数表示饼图的标题，第 2 个参数表示饼图的数据集，第 3 个参数表示饼图是否显示 legend，第 4 个参数表示饼图是否显示提示，以及第 5 个参数表示饼图是否为链接。这些参数同最后显示的 ChartFrame 对象的结构一一对应，如图 16.27 所示。

虽然 JFreeChart 对象代表的是整个饼形对象，但是如果想显示出来，就必须通过 ChartFrame 类来实现。如何创建 ChartFrame 对象呢？查看帮助文档可以发现 ChartFrame 类的构造函数：

```
ChartFrame(java.lang.String title, JFreeChart chart)
```

第 1 个参数为 ChartFrame 的标题，第 2 个参数为所要显示的 JFreeChart 对象。最后通过 ChartFrame 对象的方法 pack()和 setVisible()把 JFreeChart 对象正确显示出来。

最后，编译和运行 JfreeChartTest.java 文件，其运行结果如图 16.28 所示。

16.3.2　Java Web 项目实现饼形图形

虽然在 16.3.1 节通过 JFreeChartt 组件的相关类在 Java 项目中实现了饼形图形，但是如果想在 Java Web 项目中显示出所要求的图形，除了使用 16.3.1 节中相关的类外，还必须做

一些必要的设置。

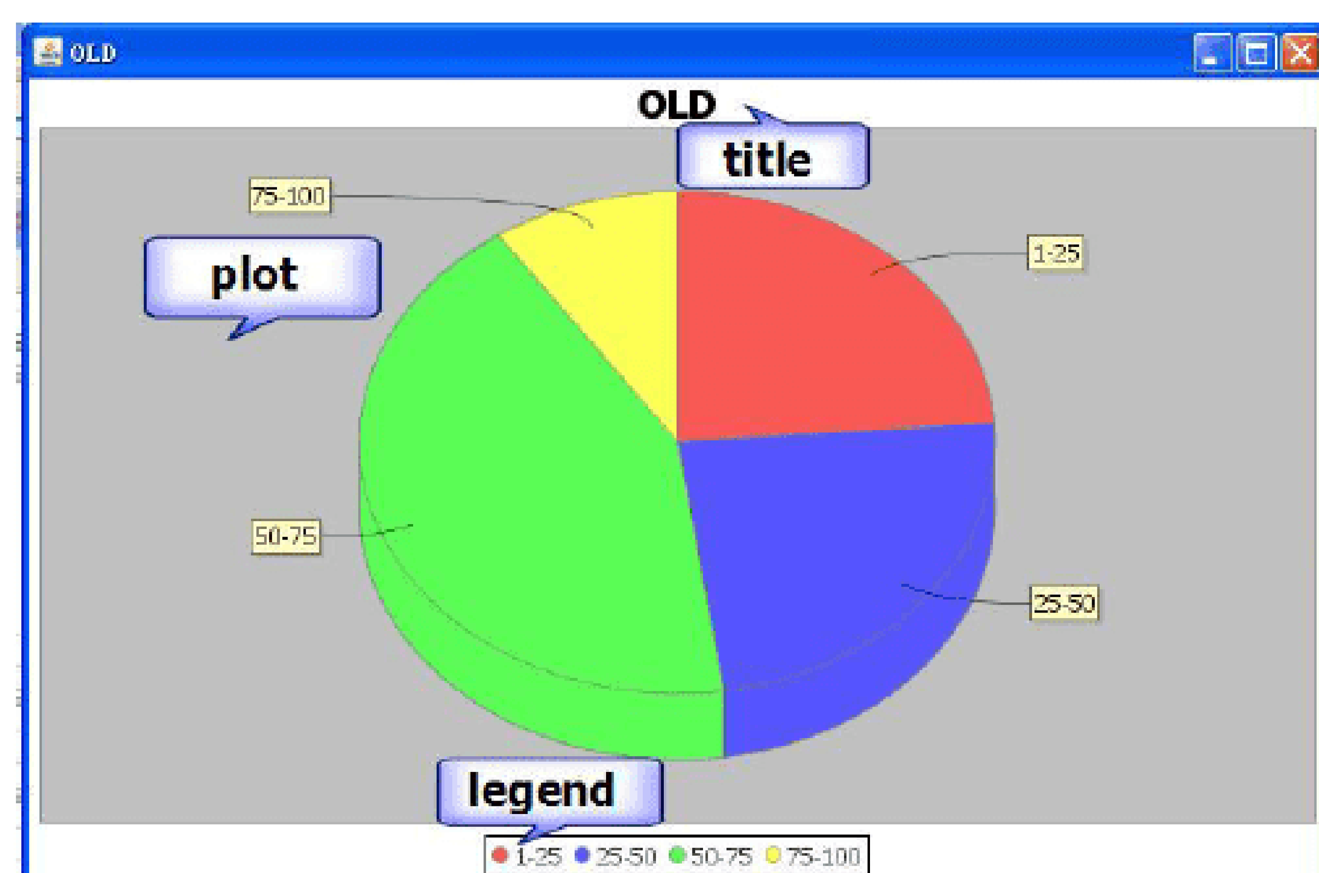


图 16.27　ChartFrame 对象结构

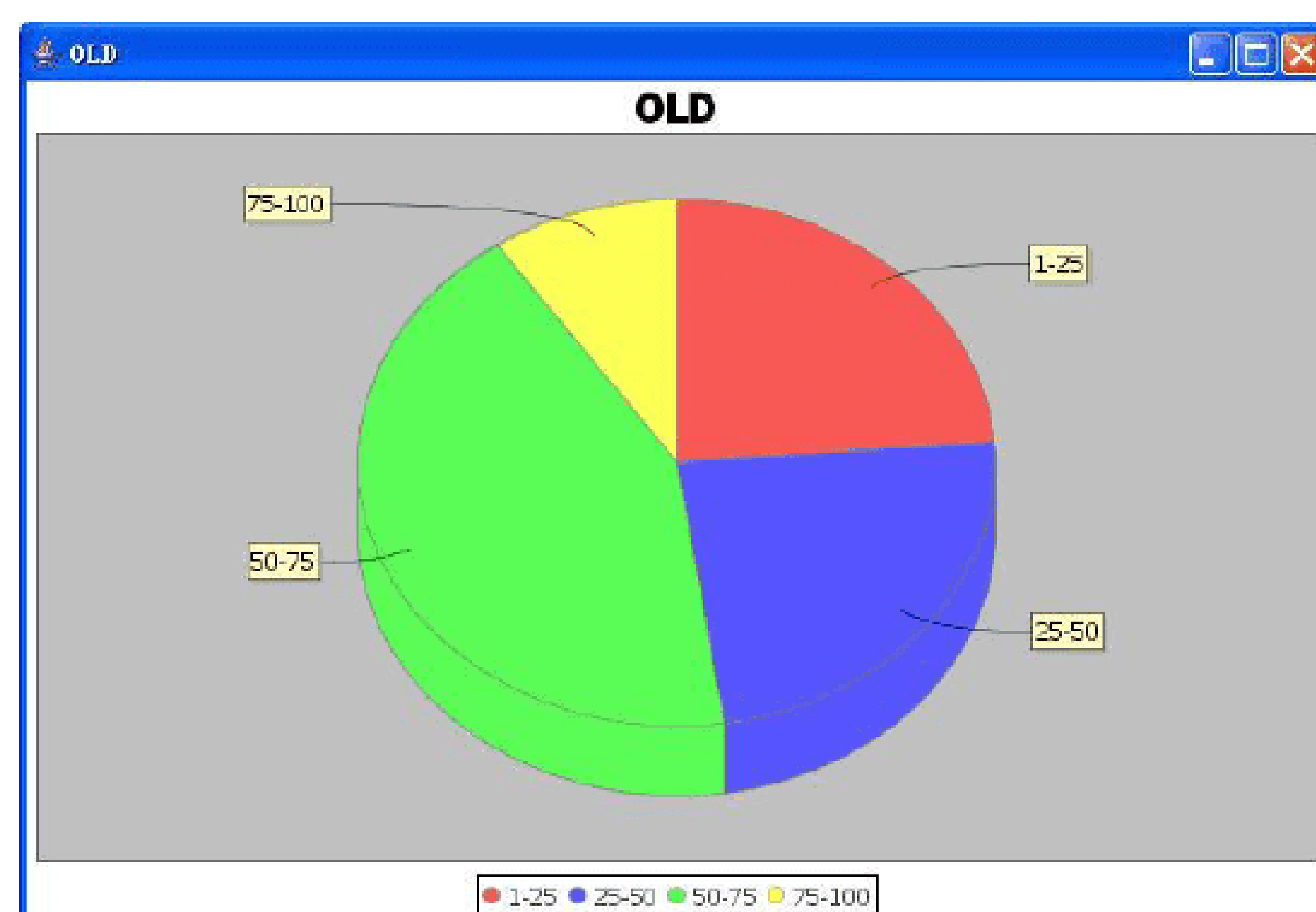


图 16.28　运行结果

下面通过实现显示饼形图形功能的 jfreecharweb 项目，讲解如何在 Java Web 项目中实现显示饼形图形。

（1）首先查看 JFreeChart 组件的帮助文档可以发现，如果想在 Java Web 项目中实现图形的显示，必须要使用 org.jfree.chart.servlet 包的相关类 DisplayChart。在使用该类时，必须要设置 web.xml 文件。代码 16.2 为 web.xml 文件增加关于 DisplayChart 这个 Servlet 程序的设置。

代码 16.2　关于 JfreeChart 组件的设置：web.xml

```
<!--配置 DisplayChart 类路径-->
<servlet>
    <servlet-name>DisplayChart</servlet-name>
    <servlet-class>
        org.jfree.chart.servlet.DisplayChart
    </servlet-class>
</servlet>
<!--配置 DisplayChart 映射路径-->
<servlet-mapping>
    <servlet-name>DisplayChart</servlet-name>
    <url-pattern>/DisplayChart</url-pattern>
</servlet-mapping>
```

【代码解析】

DisplayChart 类是服务器端的 Servlet 程序，可以实现从临时目录中把图片以流的形式发送给浏览器。该类 JFreeChart 组件已经实现，开发人员只需要在 web.xml 文件中对该类进行相关设置就可以。

（2）其次在 Java Web 项目中如果想显示图形，一般是通过图片的形式实现的。所以必须要生成关于 JFreeChart 对象的图片，然后通过标签<src>把该图片显示出来。代码 16.3 实现了生成 JFreeChart 对象的图片并显示该图片。

代码 16.3　生成和显示图片：jfreeChart.jsp

```
<!--引入相关类-->
```

```
<%@ page
    import="org.jfree.data.general.DefaultPieDataset,org.jfree.chart.Cha
rtFactory,org.jfree.chart.JFreeChart,org.jfree.chart.servlet.*"%>
...
    <body>
        <%
            //生成数据集对象
            DefaultPieDataset dpd = new DefaultPieDataset();
            dpd.setValue("1-25", 25);
            dpd.setValue("25-50", 25);
            dpd.setValue("50-75", 45);
            dpd.setValue("75-100", 10);
            //生成 JfreeChart 对象
            JFreeChart chart = ChartFactory.createPieChart3D("OLD", dpd,
            true,false, false);
            //生成图片并返回该图片的名字
            String fileName = ServletUtilities.saveChartAsPNG(chart, 800,
            600,session);
            //调用 DisplayChart 类返回 url 对象
            String url = request.getContextPath() + "/DisplayChart?
            filename="+ fileName;
        %>
        <img src="<%=url%>" width="600" height="500">
    </body>
</html>
```

【代码解析】

- 首先如果想把 JfreeChart 对象生成相对应的图片，可以通过 org.jfree.chart.servlet 包中的 ServletUtilities 类来实现。该类中的方法用来生成相对应的图片，例如：

```
static java.lang.String saveChartAsJPEG(JFreeChart chart,
                        int width, int height,
                        javax.servlet.http.HttpSession session
                        )
```

第 1 个参数表示 JfreeChart 对象，第 2 个参数和第 3 个参数分别为所要生成图片的宽度和高度，第 4 个参数表示 session 对象。该类不仅把生成的 JfreeChart 对象的图片保存到临时文件中，同时还返回该图片的名字。

- 接着当图片生成成功后，就需要通过类 DisplayChart 把该图片返回给浏览器。如何获取该类呢？可以通过 request.getContextPath()获取服务器端的上下文来实现该类的获取。由于该类需要一个返回图片的名字，所以必需要附带一个名为 filename 的参数。最后通过标签<src>把该图片显示出来。

（3）最后单击工具栏上的按钮，把该项目发布到服务器。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/jfreecharweb/jfreeChart.jsp，运行结果如图 16.29 所示。在 Tomcat 服务器中临时文件为 Tomcat 服务器安装目录\temp 文件夹，当多次刷新显示 jfreeChart.jsp 文件的页面时，会在服务器的临时文件中生成多个文件名不同的图片，如图 16.30 所示。

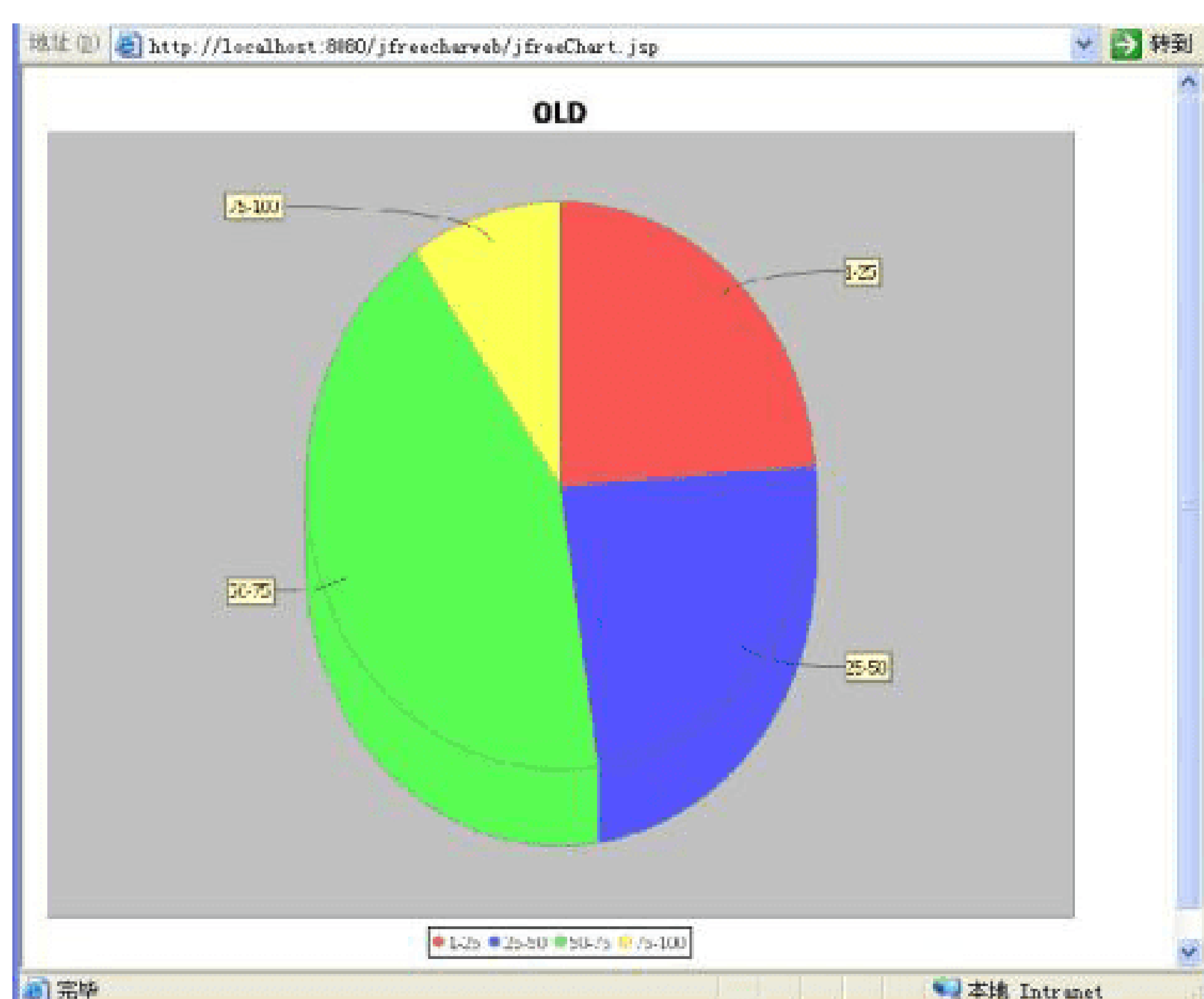


图 16.29　运行结果

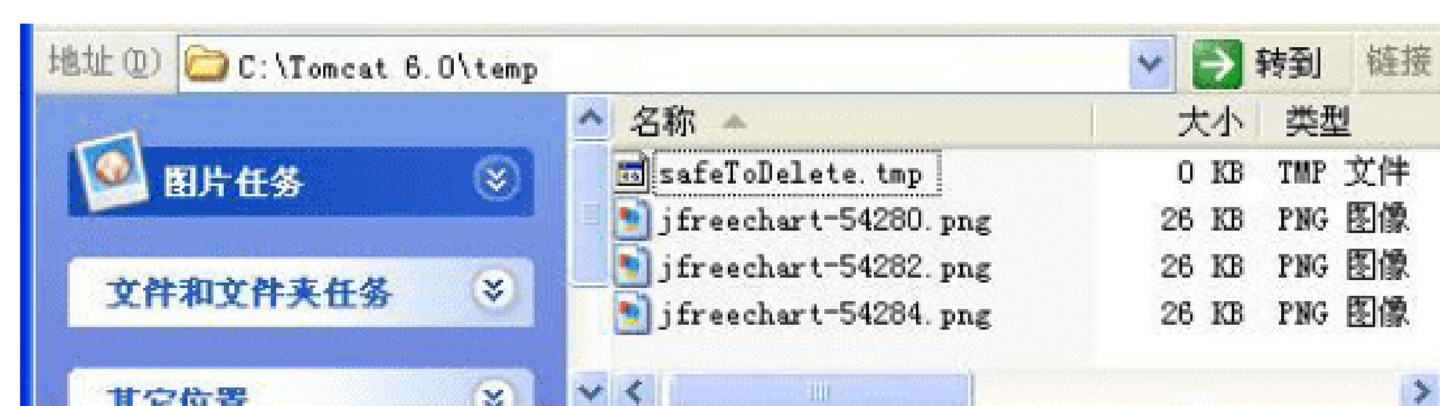


图 16.30　图形列表

16.4　实现网上投票系统

本章通过 Struts2.x+JFreeChart 框架技术来实现网上投票系统，网上投票系统程序架构如图 16.31 所示，它包含一个 JSP 页面和一个 Servlet 程序：selectvote.jsp 和 ViewResultAction.java。

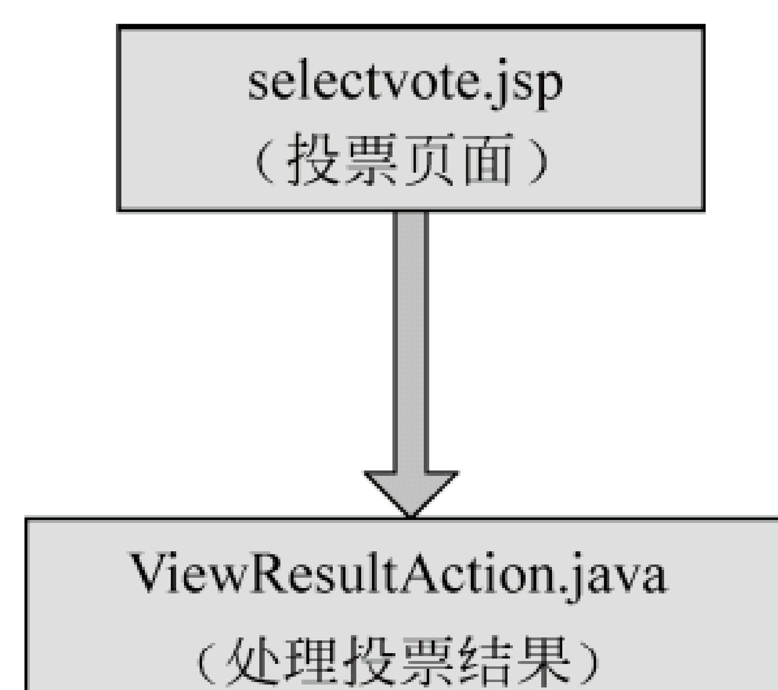


图 16.31　程序关系图

16.4.1　实现投票页面

selectvote.jsp 页面为了让浏览者实现投票，不仅需要显示出投票项目的相关信息，而且还需要把浏览者投票的结果传送给服务器。代码 16.4 演示了如何来实现投票页面。

代码 16.4　投票页面：selectvote.jsp

```
...
<body>
<h1><font color="blue">请选择喜欢的动物</font></h1>
<s:form action="viewResult">
<!--实现小狗的复选框-->
<s:checkbox  name="interest"  label=" 小   狗 "  fieldValue="xiaogou"
labelposition="left"></s:checkbox>
<!--实现小猫的复选框-->
<s:checkbox  name="interest"  label=" 小 猫 "  fieldValue="xiaomao"
labelposition="left"></s:checkbox>
<!--实现小白兔的复选框-->
<s:checkbox  name="interest"  label=" 小 白 兔 "  fieldValue="xiaobaitu"
labelposition="left"></s:checkbox>
<!--实现小猪的复选框-->
<s:checkbox  name="interest"  label=" 小    猪 "  fieldValue="xiaozu"
labelposition="left"></s:checkbox>
 <s:submit value="提交"></s:submit>
</s:form>
```

```
</body>
...
```

注意：在上述代码中实现了 4 个复选框，要注意这 4 个复选框的 name 属性是相同的，都为 interest。

16.4.2　处理投票结果

ViewResultAction 程序是服务器端的 Servlet 程序，用来实现处理投票结果的同时把投票结果利用图表的形式显示出来。该程序的具体内容如代码 16.5 所示。

代码 16.5　处理投票结构：ViewResultAction.java

```
...
public class ViewResultAction extends ActionSupport
{
    private JFreeChart chart;                    //创建 chart 属性
    private List<String> interest;               //创建 interest 属性
    //设置 chart 属性
    public JFreeChart getChart()
    {
        //创建 JfreeChart 对象
        chart = ChartFactory.createBarChart3D("兴趣统计结果","项目","结果",
        this.getDataset(), PlotOrientation.VERTICAL, false, false,false);
        //设置标题
        chart.setTitle(new TextTitle("兴趣统计结果",new Font("黑体",
        Font.BOLD,22)));
        //获取 CategoryPlot 对象
        CategoryPlot plot = (CategoryPlot)chart.getPlot();
        //获取水平轴
        CategoryAxis categoryAxis = plot.getDomainAxis();
        //设置水平轴标签的字体
        categoryAxis.setLabelFont(new Font("宋体",Font.BOLD,22));
        //设置水平轴标签的倾斜度
        categoryAxis.setCategoryLabelPositions(CategoryLabelPositions.
        UP_45);
        //获取竖轴
        NumberAxis numberAxis = (NumberAxis)plot.getRangeAxis();
        //设置竖轴标签的倾斜度
        numberAxis.setLabelFont(new Font("宋体",Font.BOLD,22));
        return chart;
    }
    //设置 interest 属性
    public List<String> getInterest()
    {
        return interest;
    }
    public void setInterest(List<String> interest)
    {
        this.interest = interest;
    }
    //编写 execute()方法
```

```
    public String execute() throws Exception
    {
        return SUCCESS;
    }
    //创建实现处理每次请求的 increaseResult()方法
    private void increaseResult(List<String> list)
    {
        ActionContext context = ActionContext.getContext();
                                            //返回 ActionContext 实例
        Map map = context.getApplication();     //获取 Application
        for (String str : list)                 //遍历传入的请求参数 list
        {
            //判断选中的选项
            if (null == map.get(str))           //当该选项为第一次时选中
            {
                map.put(str, 1);
            }
            Else                                //当该选项不是第一次时选中
            {
                map.put(str, (Integer) map.get(str) + 1);
            }
        }
    }
    //创建图像的数据集
    private CategoryDataset getDataset()
    {
        //创建数据集
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        this.increaseResult(this.getInterest());        //处理当前的请求
        ActionContext context = ActionContext.getContext();
                                            //获取 ActionContext 实例
        Map map = context.getApplication();     //获取 Application 对象
        //为数据集中添加数值
        dataset.setValue((Integer) map.get("xiaogou"), "", "小狗");
        dataset.setValue((Integer) map.get("xiaomao"), "", "小猫");
        dataset.setValue((Integer) map.get("xiaobaitu"), "", "小白兔");
        dataset.setValue((Integer) map.get("xiaozu"), "", "小猪");
        return dataset;
    }
}
```

【代码解析】

- 上述代码中首先定义了两个属性：chart 和 interest。属性 chart 主要用来获取所要显示的 JfreeChart 对象，而属性 interest 则用来存储每次请求的参数。
- increaseResult()方法用来把每次浏览者的投票结果存储起来。由于投票的总结果是存储在服务器端的 Application 对象中，所以必须要获取该对象。在 Struts 2.x 中可以先通过 ActionContext.getContext()方法获取 ActionContext 实例，然后通过该实例的 getApplication()方法获取 Application 对象。为什么返回的是 Map 集合对象呢？是因为在 Struts 2.x 中是利用 Map 对象模拟 Application 对象的。获取投票的总结果后，就可以通过遍历请求中的投票结果，把该结果添加到投票总结果中。
- 在创建数据集的 getDataset()方法中，主要是把存储在 Application 对象中的投票总结果值设置到 dataset 对象中。this.increaseResult(this.getInterest())这句代码的主要作用就是调用 increaseResult()方法把当前请求中的投票结果存储到投票总结果中。

❑ 最后在获取 JfreeChart 对象的 getChart()方法中，创建出符合投票结果的图形。

接着在 struts.xml 文件中实现对该 ViewResultAction 类进行配置。代码 16.6 实现对 ViewResultAction 类的配置。

代码 16.6　struts 配置文件：struts.xml

```
...
    <!-- 创建自己的包，该包必须继承默认包-->
    <package name="struts2" extends="jfreechart-default">
        <!-- 配置关于 ViewResultAction 的 Action-->
        <action name="viewResult"
            class="com.cjg.action.ViewResultAction">
            <result name="success" type="chart">
                <param name="height">600</param>
                <param name="width">800</param>
            </result>
        </action>
    </package>
...
```

【代码解析】

在设置<action>标签时，定义其子标签<result>的属性中出现了“type=chart”。如果在<package>标签还是设置继承为 extends="struts-default"，在启动程序时就会报错。因为查看 Struts2.x 的 struts-default.xml 文件没有关于 chart 类型的 result。

如果想查看关于 jfreechart-default 的定义，可以解压 JfreeChartPlugin 插件的 jar 包，该 jar 包的目录结构如图 16.32 所示。打开 struts-plugin.xml 文件，代码 16.7 为关于 jfreechart-default 的定义。

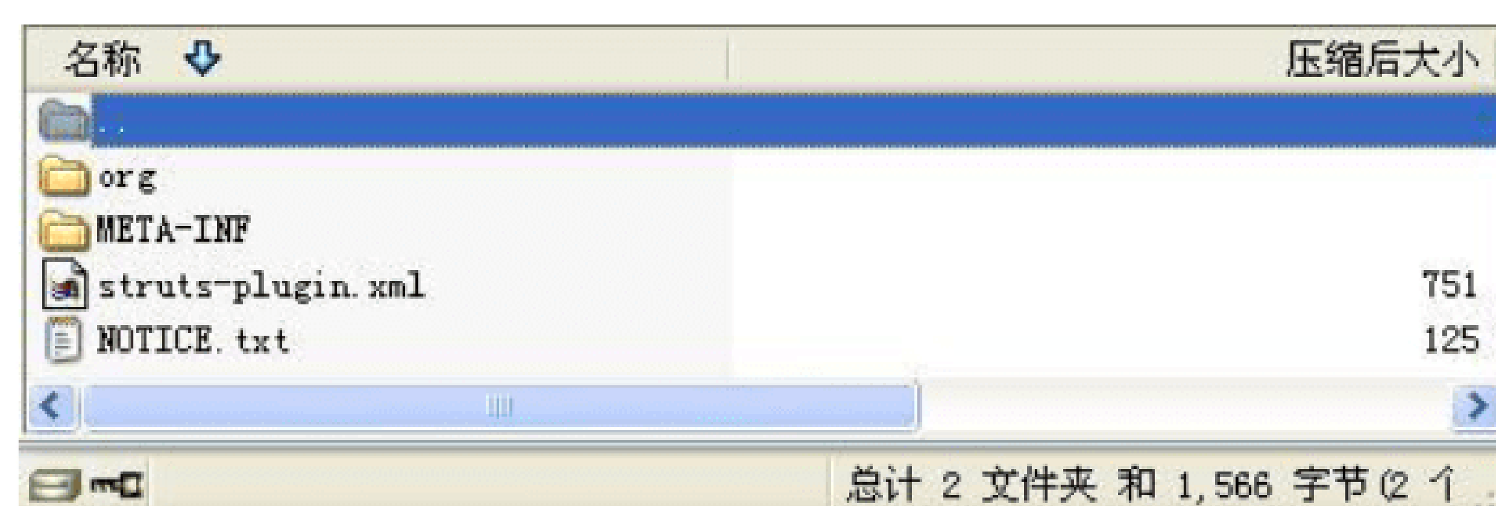


图 16.32　目录结构

代码 16.7　struts-plugin.xml 配置文件：jfreechart-default

```
<?xml version="1.0" encoding="UTF-8" ?>
...
<struts>
   <package name="jfreechart-default">
    <result-types>
        <result-type name="chart" class="org.apache.struts2.dispatcher.
        ChartResult">
            <param name="height">150</param>
            <param name="width">200</param>
        </result-type>
    </result-types>
   </package>
</struts>
```

【代码解析】

- 在上述代码中，定义名为为 chart 类型的 result 结果。
- 在具体开发时，要修改<package name="jfreechart-default">该句代码为<package name="jfreechart-default" extends="struts-default">，使得 struts.xml 文件能够使用关于 Struts2.x 的默认配置。

如果想修改 JfreeChartPlugin 插件的 jar 包，不能直接在开发环境 MyEclipse 中打开修改。首先解压该 jar 包，然后再修改相关文件，最后利用 jar 命令重新把这些文件组织成 jar 包。具体命令如图 16.33 所示。



图 16.33　jar 命令

通过 http://struts.apache.org/2.x/docs/jfreechart-plugin.html 网址打开关于 JFreeChart Plugin 插件的帮助文档。根据该文档可以发现 JFreeChart Plugin 插件的作用跟类 DisplayChart 的作用差不多，但是其功能却比类 DisplayChart 强大得多。该插件提供了一个 ChartResult 类来代替生成图片，使得程序员可以根据自己的意愿将图片输出到文件中或者其他类型的视图中。

16.5　小　　结

本章主要介绍网上投票系统，本系统基于 Struts2+JfreeChart 共同构建而成。为了让读者深入理解网上投票系统，首先通过图表组件 JfreeChart 实现关于图表的 Java 项目和 JavaBean 项目，接着通过 Struts2+JfreeChart 框架实现完整的网上投票系统。在该系统中为了更友好，利用 JfreeChart 组件的图形显示投票结果。

第 17 章　商业银行网上账户管理系统（Struts 2.x）

为了深入理解 Struts 2.x 框架，本章将通过基于 Struts 2.x 框架的具体应用——商业银行网上账户管理系统，来演示如何开发 Struts 2.x 应用系统。为了能够更加清楚地了解 Struts 2.x 框架，本章将不结合其他的开源框架，而是完全应用 Struts 2.x 框架来实现。

本章将通过 Struts 2.x 框架技术来实现一个完整的商业银行网上账户管理系统，而数据库管理系统则为微软公司（Microsoft）的 SQL Service 2000。

17.1　商业银行网上账户管理系统简述

商业银行网上账户管理系统为用户提供了账户申请、账户注销、存款、取款、查询账户余额等业务。该系统运行在 Windows 操作系统下，为了能够让用户不经过培训就能够使用，该系统提供了简洁、易用的图形界面。

17.1.1　商业银行网上账户管理系统设计原理

为了让读者可以快速理解和掌握商业银行网上账户管理系统，在具体讲解时对该系统应用最常见的 4 层模型。在整个系统中，将以 Struts 2.x 作为 MVC 框架，具体架构如图 17.1 所示，而具体目录如图 17.2 所示。

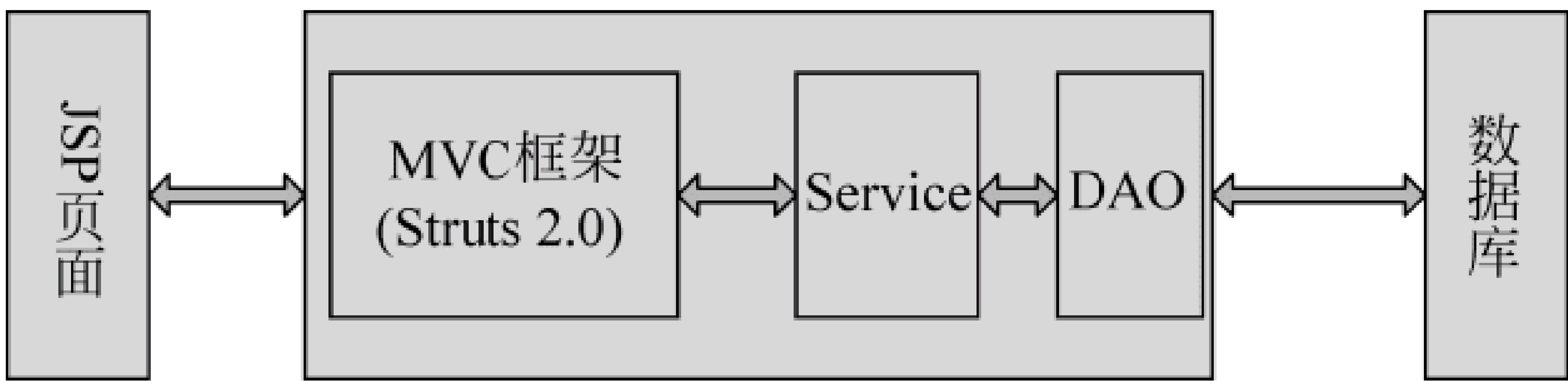


图 17.1　系统框架

所谓的四层模型，即领域模型层（POJO）、持久层、业务层和表现层，它们各自的作用如表 17.1 所示。

表 17.1　四层模型功能

模 型 层 次	作　　用
领域模型层	该层用来将数据库字段抽象化
持久层	该层用来实现数据库存取操作

续表

模 型 层 次	作　　用
业务层	该层用来实现系统的逻辑
表现层	该层不仅用来提供页面所需要的信息，而且还用来处理用户请求及访问控制

17.1.2　商业银行网上账户管理系统的基本原理

在本节中，将以直观的方式向读者介绍整个商业银行网上账户管理系统实现的功能。在该系统中存在两种用户，他们各自可以实现的功能也不尽相同，如图 17.3 所示。

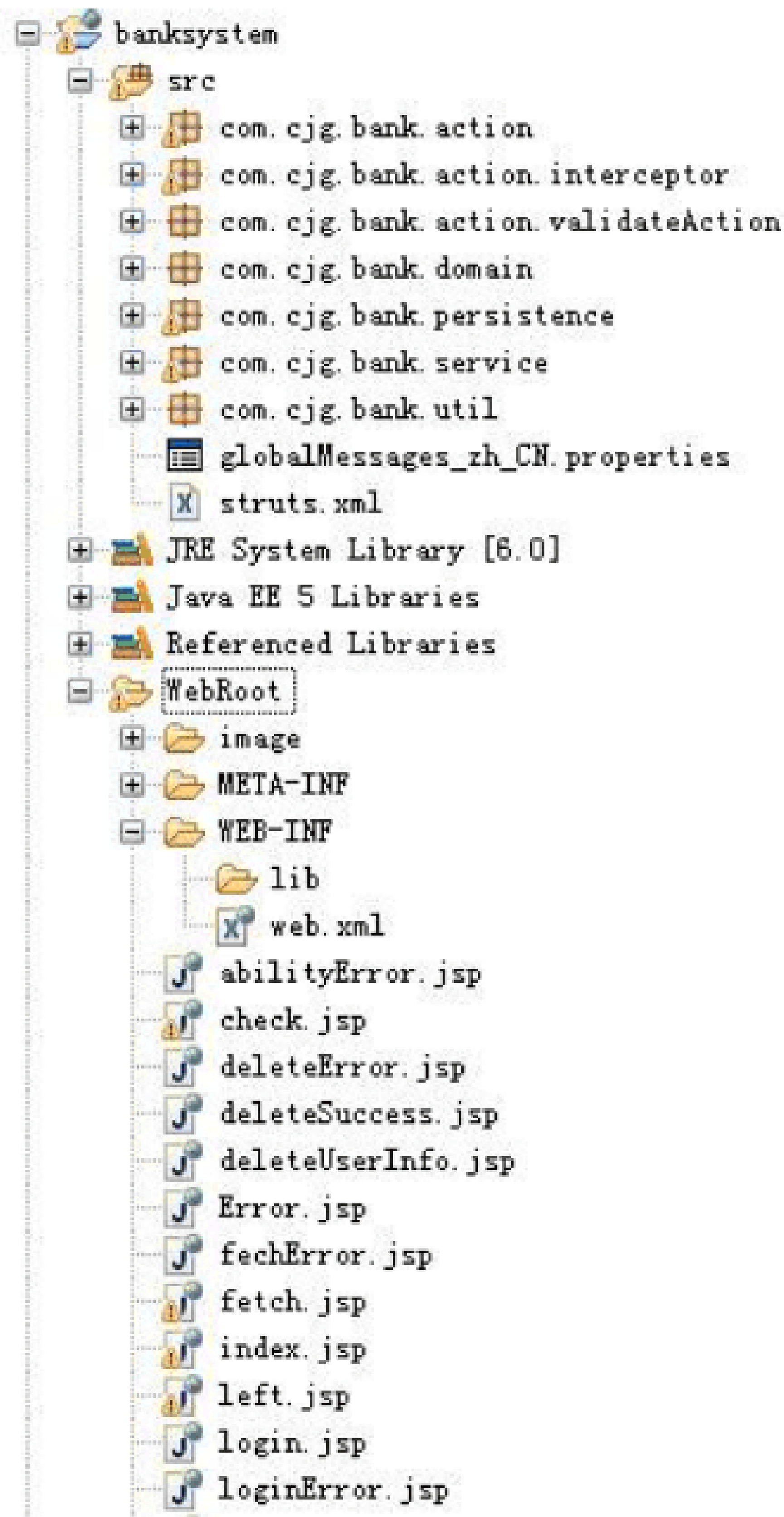


图 17.2　项目目录

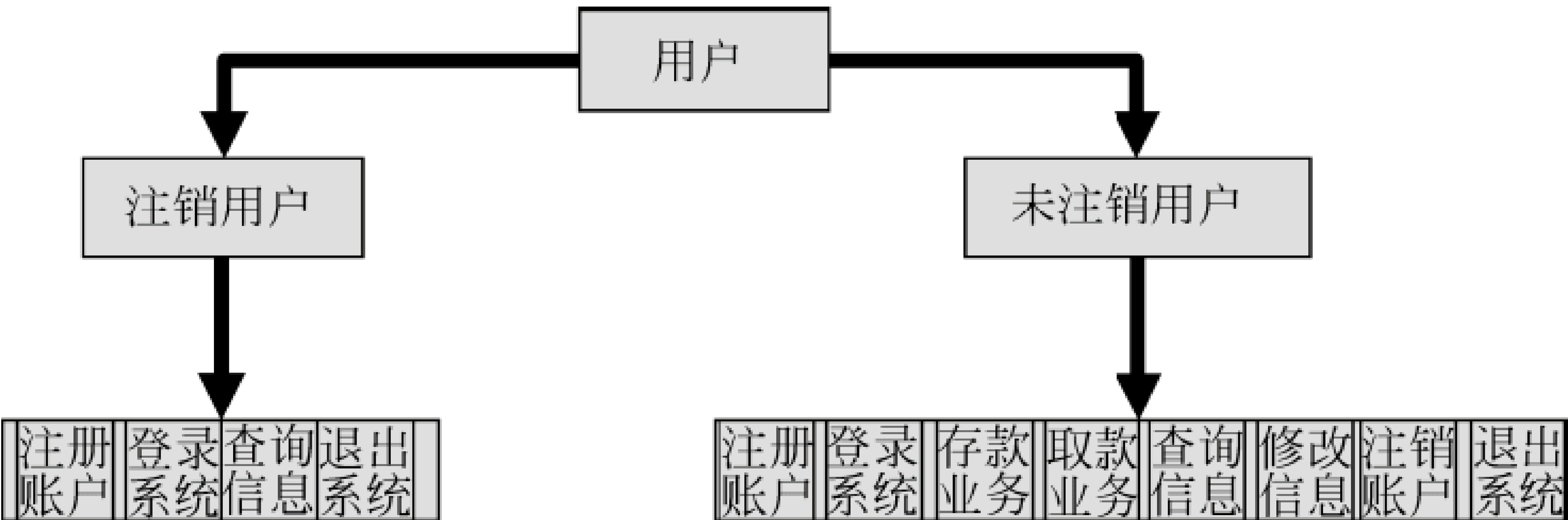


图 17.3　不同用户使用功能

1．注册新账户

任何用户都可以通过单击登录页面（如图 17.4 所示）中的“注册新账户”链接打开注册新账户页面（如图 17.5 所示）。在该页面中输入相应信息并提交就会转到显示新账户信息的页面（如图 17.6 所示）。

2．登录系统

拥有账户的用户通过网址 http://localhost:8088/banksystem/login.jsp 打开登录页面（如图 17.7 所示），在该页面中输入自己的账号和密码并提交就可以转入系统主界面（如图 17.8 所示），否则会转入出错页面（如图 17.9 所示）。

3．查询账户信息

当用户通过自己的账户和密码进入系统主界面后，通过单击“个人信息”按钮就可以打开关于该用户信息页面（如图 17.10 所示）。



图 17.4　登录页面



图 17.5　注册页面



图 17.6　新账户信息页面



图 17.7　登录页面

4．修改账户信息

当用户通过自己的账户和密码进入系统主界面后，通过单击“更改信息”按钮就可以打开更改用户信息页面（如图 17.11 所示）。在该页面中输入新的账户信息并提交，就转到更新后的账户信息页面（如图 17.12 所示）。

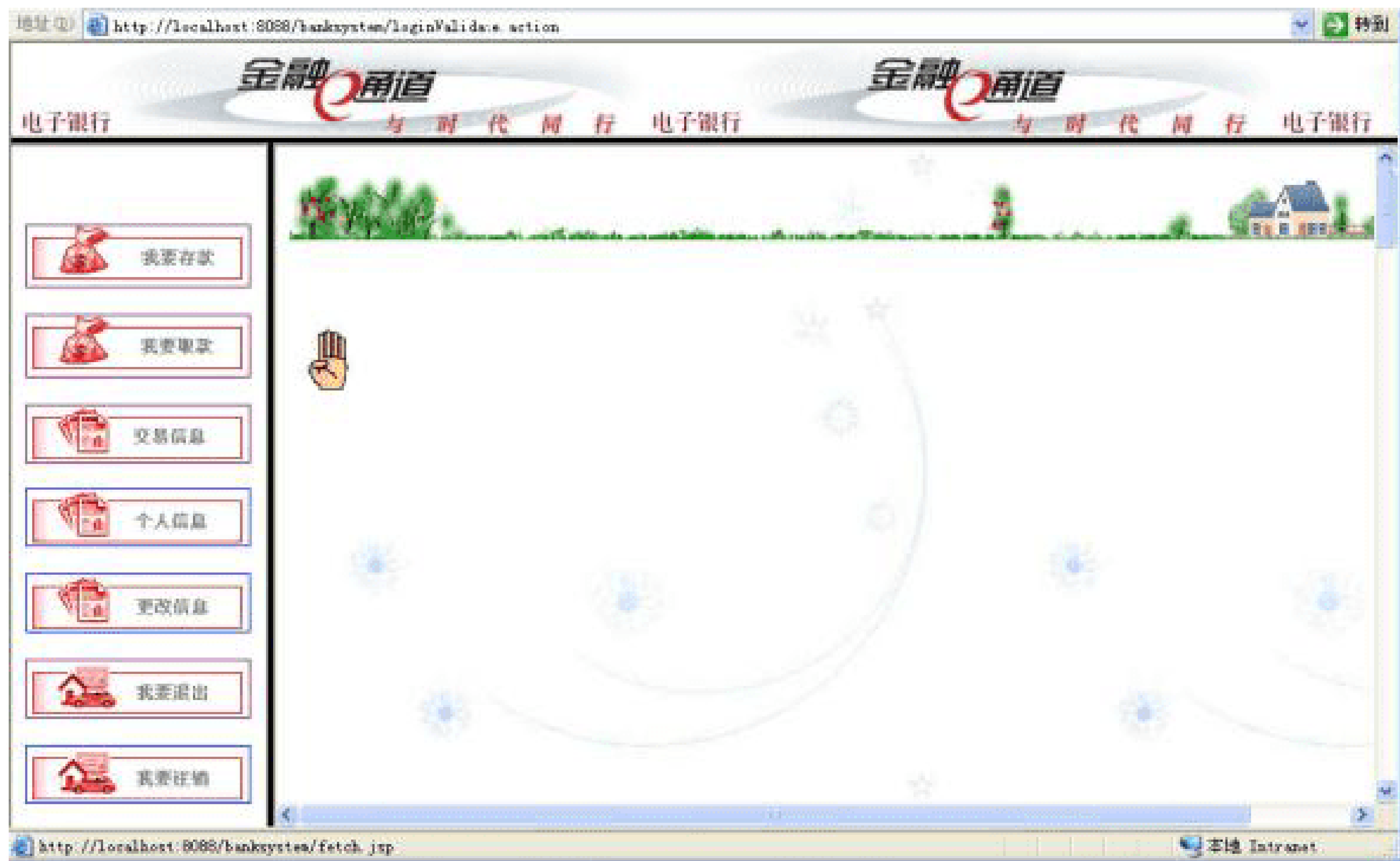


图 17.8　系统主界面





图 17.9　登录失败页面　　　　图 17.10　查看账户信息



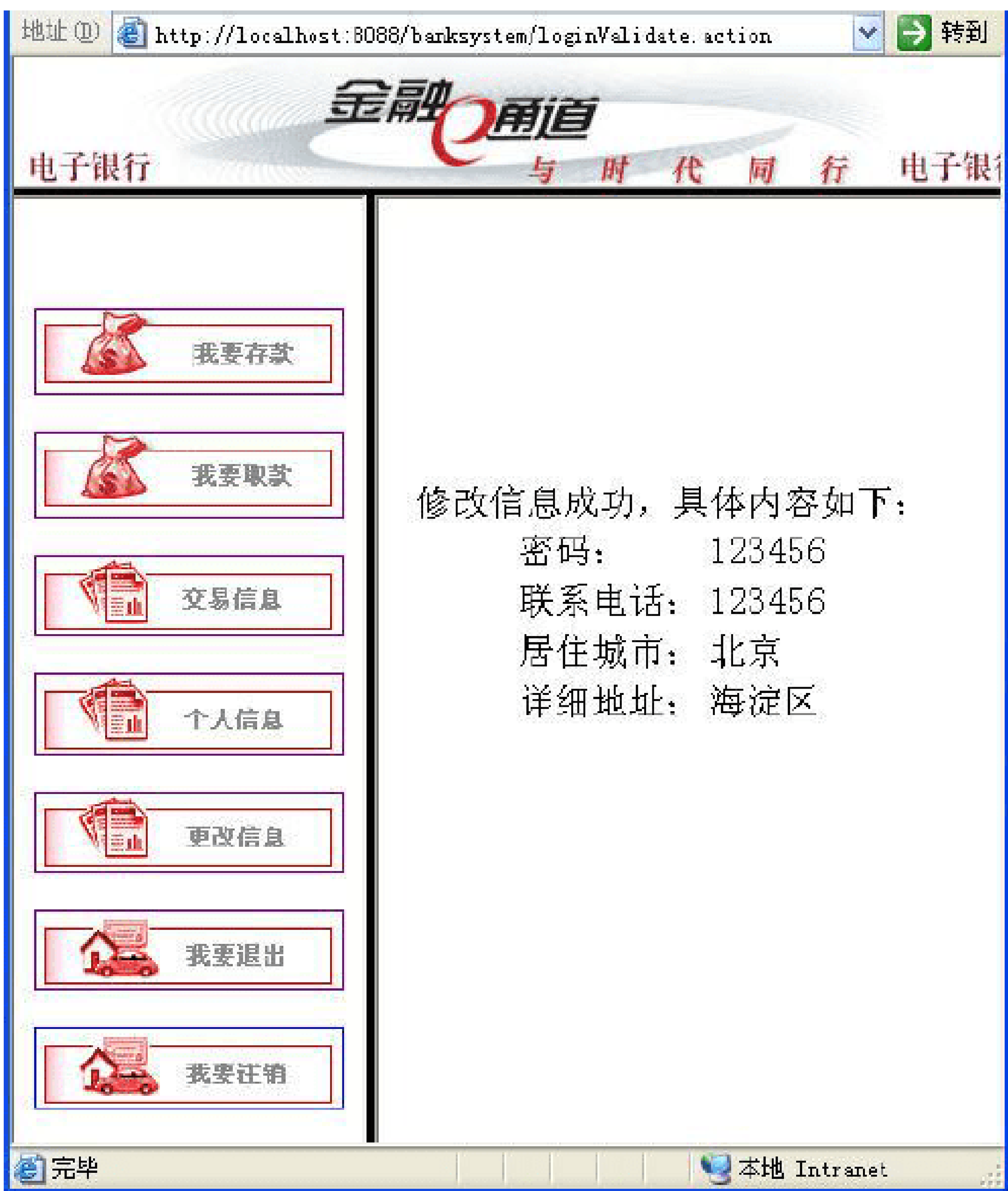


图 17.11　更改用户信息页面　　　　图 17.12　更新后账户信息

5．存款

当用户通过自己的账户和密码进入系统主界面后，通过单击“我要存款”按钮就可以打开存款相关页面（如图 17.13 所示）。在该页面中输入存款金额并提交后就会转到显示该账户余额的界面（如图 17.14 所示）。



图 17.13　存款页面



图 17.14　显示账户余额

6．取款

当用户通过自己的账户和密码进入系统主界面后，通过单击“我要取款”按钮就打开取款相关页面（如图 17.15 所示）。在该页面中输入取款金额并提交后就会转到显示该账户余额的界面（如图 17.16 所示）。



图 17.15　取款页面



图 17.16　余额页面

7．交易信息

当用户通过自己的账户和密码进入系统主界面后，通过单击“交易信息”按钮就转到关于交易信息的页面（如图 17.17 所示）。



图 17.17　交易信息页面

8．注销用户

当用户通过自己的账户和密码进入系统主界面后，通过单击“我要注销”按钮就会转到关于注销的页面（如图 17.18 所示）。



图 17.18　注销用户页面

9．退出

当用户通过自己的账户和密码进入系统主界面后，通过单击“我要退出”按钮就会转

到成功退出系统页面（如图 17.19 所示）。



图 17.19　退出系统页面

17.2　商业银行网上账户管理系统前期准备

本节除了将详细地介绍如何设计关于商业银行网上账户管理系统的数据库和表格外，还将配置实现该系统的 Struts 2.x+SQL Server 框架的环境。其中 Struts 2.x 框架的版本号为 Struts 2.0，数据库 Microsoft SQL Server 为 SQL Server 2000。

17.2.1　设计数据库

商业银行网上账户管理系统需要建立一个数据库并在该数据库中建立两张表，即存放表的数据库 banksystem、存放用户信息的表 userinfo 和存放交易信息的表 trader。

1．创建数据库banksystem

如果想创建出数据库 banksystem，可以在 SQL Server 2000 的查询分析器窗口中输入如下命令：

```
CREATE DATABASE 'banksystem'
```

2．创建表格userinfo

如果想创建出表 userinfo，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表的具体信息如表 17.2 所示。

表 17.2　表useinfo信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	Int(10)	编号
userName	varchar(50)	用户名
Userflag	Int(10)	账户注销标记
userAge	Int(10)	用户年龄
idCard	varchar(50)	身份证号
Tel	varchar(50)	联系电话

续表

字 段 名 称	数 据 类 型	字 段 说 明
City	varchar(50)	居住城市
userAddress	varchar(50)	详细地址
userSex	varchar(50)	性别
userNo	varchar(50)	用户账号
Balance	Int(10)	账户余额
Userflag	Int(10)	注销标志

实现用户信息模型的内容如代码 17.1 所示。

代码 17.1　用户信息模型：Userinfo.java

```
...
public class UserInfo {
    private String userName;                    //创建用户名属性
    private int userAge;                        //创建用户年龄属性
    private String idCard;                      //创建身份证号属性
    private String password;                    //创建密码属性
    private int id;                             //创建编号属性
    private String userSex;                     //创建性别属性
    private String tel;                         //创建电话号码属性
    private String address;                     //创建地址属性
    private String city;                        //创建城市属性
    private String userNO;                      //创建用户账号属性
    private int balance = 0;                    //创建账户余额属性
    private int userflag = 0;                   //创建注销标志属性
    //省略属性 userName、userAge、idCard、password、id、userSex、tel、address、
    city、userNO、balance 和 userflag 的 set()和 get()方法
...
}
```

3. 创建表trader

如果想创建出表 trader，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表的具体信息如表 17.3 所示。

表 17.3　表trader信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	int(10)	编号
Trade	varchar(50)	交易类型
Balance	int(10)	余额
dataTime	Varchar(50)	交易时间
userNo	Varchar(50)	用户账户
Money	int(10)	交易金额

实现交易模型的内容如代码 17.2 所示。

代码 17.2　交易模型：TradeInfo.java

```
...
public class TradeInfo {
```

```
    private String datatime;                          //创建交易时间属性
    private String userNO;                            //创建用户账户属性
    private int money;                                //创建交易金额属性
    private int balance = 0;                          //创建余额属性
    private int id;                                   //创建编号属性
    private String trade;                             //创建交易类型属性
    //省略属性 datatime、userNO、money、balance、id 和 trade 的 set()和 get()
    方法
...
}
```

17.2.2　关于 Struts 2.0 的准备

由于该项目完全是由 Struts 2.x 框架来实现，所以对于 Struts 2.x 框架除了需要引入相应的 jar 包外，还必须得对 struts.xml 和 web.xml 文件做相应的配置。

1．引入jar包

引入关于 Struts 2.0 框架的核心包：struts2-core-2.0.11.jar、xwork-2.0.4.jar、ognl-2.6.11.jar、freemarker-2.3.8.jar 和 commons-logging-1.0.4.jar。最后由于需要连接数据库 SQL Server 2000，所以还需要引入关于该数据库的驱动。

注意：前 5 个 jar 包在 Struts 2.0 框架的 jar 包中就可以找到，而最后一个关于数据库的驱动则必须从该数据库的官方网站下载。

2．修改web.xml文件

为了使商业银行网上账户管理系统支持 Struts 2.0 框架，需要在 web.xml 文件中增加如代码 17.3 所示的内容。

代码 17.3　修改 web.xml 文件：web.xml

```
...
    <!--设置过滤器类-->
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <!--设置过滤器映射路径-->
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
...
```

3．创建struts.xml文件

为了使商业银行网上账户管理系统支持 Struts 2.0 框架，还需要在 banksystem\src 目录

中创建 struts.xml 文件，该文件的内容如代码 17.4 所示。

代码 17.4　实现 struts.xml 文件：struts.xml

```
...
<struts>
    <constant name="struts.i18n.encoding" value="GBK"/>
    <constant          name="struts.custom.i18n.resources"         value=
    "globalMessages"/>
    <package name="default" extends="struts-default">
...
    </package>
</struts>
```

至此，就完成了对商业银行网上账户管理系统中 Struts 2.0 框架的配置。

17.3　商业银行网上账户管理系统具体实现——持久层

为了让读者可以快速地理解和掌握商业银行网上账户管理系统，在具体讲解时对该系统应用最常见的 4 层模型：领域模型层、持久层、业务层和表现层。由于 17.2 节已经讲解了关于两个数据库表的领域模型层，所以本节将详细介绍关于两个数据库表的持久层。

17.3.1　关于 userInfo 表操作

在实现关于 userInfo 关联表操作时，采用了 DAO 模式。本节将实现对表格 userInfo 操作：注册业务、修改个人信息业务和注销业务等

实现操作表格 userInfo 功能的接口如代码 17.5 所示。

代码 17.5　操作 userInfo 接口：UserDAO.java

```
...
public interface UserDAO {
    boolean login(UserInfo user) throws SQLException;          //登录功能
    void registService(UserInfo user) throws SQLException;     //注册功能
    UserInfo selectUser(String userNO) throws SQLException;
                                                         //查询个人信息功能
    //修改个人信息功能
    void updateUserInfo(UserInfo user,String userNO) throws SQLException;
    void deleteUserInfo(String userNO) throws SQLException;
                                                         //注销个人账户功能
}
```

【代码解析】

- login()方法用来实现账号的登录。
- registService()方法用来实现账户的注册。
- selectUser()方法用来实现查询账户信息。
- updateUserInfo()方法用来实现修改账户信息。
- deleteUserInfo()方法用来实现注销账户。

实现关于表格 userInfo 接口 UserDAO 的类如代码 17.6 所示。

代码 17.6　操作 userInfo 实现类：UserDAOImpl.java

```
...
public class UserDAOImpl implements UserDAO {
    Connection conn = null;                          //创建变量 conn
    Statement st = null;                             //创建变量 st
    //登录系统功能
    public boolean login(UserInfo user) throws SQLException {
        boolean flag = false;                        //创建 boolean 类型变量
        MD5 md5 = new MD5();                         //生成工具类 MD5 的对象
        String userNO = user.getUserNO();            //获取用户账号
        //获取工具类对象 md5 处理后的用户密码
        String password = md5.getMD5ofStr(user.getPassword());
        try {
            conn = DBConnection.getDBC();
                                      //通过工具类 DBConnection 获取连接信息
            st = conn.createStatement();             //为变量 st 赋值
            //创建 SQL 语句
            String sql = "select userNO from userInfo where userNO='" + userNO
                    + "' and password = '" + password + "'";
            ResultSet rs = st.executeQuery(sql);     //SQL 语句执行结果
            //判断执行结果 rs
            if (rs.next()) {                         //登录成功
                flag = true;
                rs.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            st.close();
            conn.close();
        }
        return flag;
    }
    //注册新用户功能
    public void registService(UserInfo user) throws SQLException {
        MD5 md5 = new MD5();                         //创建工具类 MD5 对象 md5
        long s = System.currentTimeMillis();         //获取系统当前时间
        String a = String.valueOf(s);                //变量 s 的类型变成 String
        user.setUserNO(a);                           //设置用户账号信息
        String sql = "insert into userInfo"          //创建 SQL 语句
                + "(userName,password,userAge,idCard,userSex,tel,city,
                userAddress,userNO,balance,userflag) "
                + "values('" + user.getUserName() + "','" + "'"
                + md5.getMD5ofStr(user.getPassword()) + "','" + "'"
                + user.getUserAge() + "','" + "'" + user.getIdCard() + "','"
                + "'" + user.getUserSex() + "','" + "'" + user.getTel() + "','"
                + "'" + user.getCity() + "','" + "'" + user.getAddress() +
                "','"
                + "'" + user.getUserNO() + "','" + 0 + "','" + 0 + "')";
        try {
            conn = DBConnection.getDBC();
            conn.setAutoCommit(false);
            st = conn.createStatement();
            st.executeUpdate(sql);                               //更新数据库
```

```
            conn.commit();
        } catch (Exception e) {
            e.printStackTrace();
            conn.rollback();
        } finally {
            st.close();
            conn.close();
        }
    }
    //查询个人账户信息功能
    public UserInfo selectUser(String userNO) throws SQLException {
        ResultSet rs = null;                        //创建 ResultSet 对象
        UserInfo userInfo = null;                   //创建用户信息对象
        //创建 SQL 语句
        String sql = "select * from userInfo where userNO = '" + userNO +
        "'";
        try {
            conn = DBConnection.getDBC();
            st = conn.createStatement();
            userInfo = new UserInfo();
            rs = st.executeQuery(sql);              //执行 SQL 语句
            if (rs.next()) {
                //把查询结果中的信息赋予对象 userInfo
                userInfo.setUserName(rs.getString("userName"));
                userInfo.setUserNO(rs.getString("userNO"));
                userInfo.setUserAge(rs.getInt("userAge"));
                userInfo.setIdCard(rs.getString("idCard"));
                userInfo.setTel(rs.getString("tel"));
                userInfo.setCity(rs.getString("city"));
                userInfo.setBalance(rs.getInt("balance"));
                userInfo.setAddress(rs.getString("userAddress"));
                userInfo.setUserSex(rs.getString("userSex"));
                userInfo.setUserflag(rs.getInt("userflag"));
                return userInfo;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            rs.close();
            st.close();
            conn.close();
        }
        return null;
    }
    //修改个人账户信息功能
    public void updateUserInfo(UserInfo user, String userNO)
            throws SQLException {
        MD5 md5 = new MD5();                    //创建工具类 MD5 对象 md5
        //创建 SQL 语句
        String sql = "update userInfo set tel ='" + user.getTel()
                + "',password='" + md5.getMD5ofStr(user.getPassword())
                + "',city='" + user.getCity() + "',userAddress='"
                + user.getAddress() + "'";
        try {
            conn = DBConnection.getDBC();
            conn.setAutoCommit(false);
            st = conn.createStatement();
            st.executeUpdate(sql);              //更新数据库
```

```
            conn.commit();
        } catch (Exception e) {
            e.printStackTrace();
            conn.rollback();
        } finally {
            st.close();
            conn.close();
        }
    }
    //注销个人信息功能
    public void deleteUserInfo(String userNO) throws SQLException {
        //创建 SQL 语句
        String sql = "update userInfo set userflag='1' where userNO='" +
        userNO
                + "'";
        try {
            conn = DBConnection.getDBC();
            conn.setAutoCommit(false);
            st = conn.createStatement();
            st.executeUpdate(sql);                    //更新数据
            conn.commit();                            //实现事务提交
        } catch (Exception e) {
            e.printStackTrace();
            conn.rollback();                          //实现事务回滚
        } finally {
            st.close();
            conn.close();
        }
    }
}
```

注意： 在上述代码中，分别实现了 UserDAO 接口中的所有方法。

17.3.2　关于 trade 表操作

在实现关于 trade 表操作时，采用了 DAO 模式。本节将实现对关联表 trade 操作：存款功能、取款功能和查询余额等功能。

实现操作表 trade 功能的接口如代码 17.7 所示。

代码 17.7　操作 trade 接口：TradeDAO.java

```
...
public interface TradeDAO {
    void saveMoney(TradeInfo tradeInfo)throws SQLException;    //存款方法
    void fetchMoney(TradeInfo tradeInfo)throws SQLException;   //取款方法
    Integer selectBalance(String userNO) throws SQLException;
                                                     //查询账户余额方法
    List selectTradeInfo(String userNO) throws SQLException;
                                               //获取用户所有交易信息方法
}
```

【代码解析】

- saveMoney()方法用来实现存款功能。
- fetchMoney ()方法用来实现取款功能。

- selectBalance()方法用来实现查询账户余额功能。
- selectTradeInfo()方法用来查询当前登录用户交易信息。

实现关于表格 trade 接口 TradeDAO 的类如代码 17.8 所示。

代码 17.8　操作 trade 实现类：TradeDAOImpl.java

```
...
public class TradeDAOImpl implements TradeDAO {
    Connection conn = null;                          //创建 Connection 类型变量
    Statement st = null;                             //创建 Statement 类型变量
    PreparedStatement psmt1 = null;          //创建 PreparedStatement 类型变量
    ResultSet rs = null;                             //创建 ResultSet 类型变量
        //实现取款功能方法
    public void fetchMoney(TradeInfo tradeInfo) throws SQLException {
        //账户余额
        int sum = tradeInfo.getBalance() - tradeInfo.getMoney();
        tradeInfo.setTrade("取款");                  //设置交易类型
        //创建 SQL 语句
        String sql = "update userInfo set balance = '" + sum
                + "' where userNO = '" + tradeInfo.getUserNO() + "'";
        Date date = Calendar.getInstance().getTime();  //获取当前时间
        //创建时间格式
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd
        HH:mm:ss");
        String dateString = formatter.format(date);    //设置时间格式
        //创建 SQL 语句
        String sql1 = "insert into trader(userNO,money,trade,balance,
        datatime) "
            + "values('"
            + tradeInfo.getUserNO()
            + "','"
            + tradeInfo.getMoney()
            + "','"
            + tradeInfo.getTrade()
            + "','"
            + sum
            + "'," + "'" + dateString + "')";
        //更新数据库
        try {
            conn = DBConnection.getDBC();            //获取数据库连接
            conn.setAutoCommit(false);
            st = conn.createStatement();             //获取陈述对象
            st.executeUpdate(sql1);                  //实现相关数据库更新
            st.executeUpdate(sql);
            conn.commit();                           //实现更新
        } catch (SQLException e) {
            e.printStackTrace();
            conn.rollback();                         //实现回滚
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            conn.setAutoCommit(true);
            st.close();                              //关闭
            conn.close();                            //关闭连接
        }
    }
```

```
    //实现存款功能方法
    public void saveMoney(TradeInfo tradeInfo) throws SQLException {
        int sum = tradeInfo.getMoney() + tradeInfo.getBalance();
                                                        //获取账户余额
        tradeInfo.setTrade("存款");                     //设置交易类型
        //创建 SQL 语句
        String sql = "update userInfo set balance = '" + sum
                + "' where userNO = '" + tradeInfo.getUserNO() + "'";
        Date date = Calendar.getInstance().getTime();  //获取当前时间
        //创建时间格式
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd
        HH:mm:ss");
        String dateString = formatter.format(date);  //设置变量 date 的格式
        //创建 SQL 语句
        String sql1 = "insert into trader(userNO,money,trade,balance,
        datatime) "
                + "values('"
                + tradeInfo.getUserNO()
                + "','"
                + tradeInfo.getMoney()
                + "','"
                + tradeInfo.getTrade()
                + "','"
                + sum
                + "'," + "'" + dateString + "')";
        //更新数据库
        try {
            conn = DBConnection.getDBC();               //获取数据库连接
            conn.setAutoCommit(false);
            st = conn.createStatement();                //获取陈述对象
            st.executeUpdate(sql1);
            st.executeUpdate(sql);
            conn.commit();                              //实现提交
        } catch (SQLException e) {
            e.printStackTrace();
            conn.rollback();                            //实现回滚
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            conn.setAutoCommit(true);
            st.close();
            conn.close();                               //关闭连接
        }
    }
    //实现查询方法
    public Integer selectBalance(String userNO) throws SQLException {
        Integer balance = new Integer(-1);              //创建余额变量
        //SQL 语句变量
        String sql = "select balance from userInfo where userNO = '" + userNO
                + " '";
        try {
            conn = DBConnection.getDBC();               //获取数据库连接
            st = conn.createStatement();                //获取陈述对象
            rs = st.executeQuery(sql);
            if (rs.next()) {
                balance = Integer.valueOf(rs.getString("balance"));
            }
        } catch (SQLException e) {
```

```
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        rs.close();                                   //关闭连接
        conn.close();
        return balance;
    }
    //实现查询余额功能方法
    public List selectTradeInfo(String userNO) throws SQLException {
        List list = new ArrayList();                  //创建 List 类型变量
        //创建 SQL 语句
        String sql = "select * from trader where userNO='" + userNO + "'";
        //查询数据库
        try {
            conn = DBConnection.getDBC();             //获取数据库连接
            st = conn.createStatement();              //获取陈述对象
            rs = st.executeQuery(sql);                //执行 SQL 语句
            while (rs.next()) {
                TradeInfo tradeInfo = new TradeInfo();
                                                  //获取交易信息对象 tradeInfo
                //设置交易信息对象的各种值
                tradeInfo.setTrade(rs.getString("trade"));
                tradeInfo.setBalance(rs.getInt("balance"));
                tradeInfo.setDatatime(rs.getString("dataTime"));
                tradeInfo.setMoney(rs.getInt("money"));
                list.add(tradeInfo);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        rs.close();
        st.close();
        conn.close();                                 //关闭连接
        return list;
    }
}
```

注意： 在上述代码中，分别实现了 TradeDAO 接口中的所有方法。

17.4　商业银行网上账户管理系统具体实现——业务层

为了让读者可以快速理解和掌握商业银行网上账户管理系统，在具体讲解时对该系统应用最常见的 4 层模型：领域模型层、持久层、业务层和表现层。本章将接着 17.3 节持久层的内容继续讲解该系统的业务层。

17.4.1　关于数据库表 userinfo 的业务层

在设计实现关于 userinfo 表操作的业务层时，采用了 DAO 模式。本节将在业务层实现对表 userinfo 操作：登录业务、注册业务、查询账户信息业务等。

业务层实现操作表 userinfo 功能的接口如代码 17.9 所示。

代码 17.9　操作表 userinfo 接口：UserFacade.java

```
...
public interface UserFacade {
    boolean login(UserInfo user) throws SQLException;      //实现登录功能
    void registService(UserInfo user) throws SQLException; //实现注册功能
    UserInfo selectUser(String userNO) throws SQLException;
                                            //实现查询个人账户信息功能
    //实现修改个人账户信息功能
    void updateUserInfo(UserInfo user, String userNO) throws SQLException;
    void deleteUserInfo(String userNO) throws SQLException;
                                            //实现注销个人账户功能
}
```

【代码解析】

- ❑ login()方法用来实现登录业务。
- ❑ registService()方法用来实现账户注册业务。
- ❑ selectUser()方法用来实现查询账户信息业务。
- ❑ updateUserInfo()方法用来实现修改账户信息业务。
- ❑ deleteUserInfo()方法用来实现注销用户账户业务。

业务层实现关于模块表 userinfo 接口 UserFacade 的类，如代码 17.10 所示。

代码 17.10　操作 userinfo 实现类：UserFacadeImpl.java

```
...
public class UserFacadeImpl implements UserFacade {
    private UserDAO userDAO;                        //创建 UserDAO 类型对象
    public UserFacadeImpl() {                       //创建构造函数
        userDAO = new UserDAOImpl();
    }
    public void deleteUserInfo(String userNO) throws SQLException {
                                                    //注销用户业务
        userDAO.deleteUserInfo(userNO);
    }
    public boolean login(UserInfo user) throws SQLException { //登录业务
        return userDAO.login(user);
    }
    public void registService(UserInfo user) throws SQLException {
                                                    //注册用户业务
        userDAO.registService(user);
    }
    //查询用户信息业务
    public UserInfo selectUser(String userNO) throws SQLException {
        return userDAO.selectUser(userNO);
    }
    public void updateUserInfo(UserInfo user, String userNO)
                                                    //更新用户信息业务
            throws SQLException {
        userDAO.updateUserInfo(user, userNO);
    }
}
```

注意：在上述代码中，分别实现了 UserFacade 接口中的所有方法。

17.4.2　关于数据库表 trade 的业务层

在设计实现关于 trade 表操作的业务层时，采用了 DAO 模式。本节将在业务层实现对表 trade 的操作：浏览模块、增加模块、删除模块和修改模块。

业务层实现操作表 trade 功能的接口如代码 17.11 所示。

代码 17.11　操作表 trade 接口：TradeFacade.java

```
...
public interface TradeFacade {
    void saveMoney(TradeInfo tradeInfo) throws SQLException;//实现存款功能
    void fetchMoney(TradeInfo tradeInfo) throws SQLException;
                                            //实现取款功能
    Integer selectBalance(String userNO) throws SQLException;
                                            //实现查询账户余额功能
    List selectTradeInfo(String userNO) throws SQLException;
                                            //实现查看用户交易信息功能
}
```

【代码解析】

- saveMoney()方法用来实现存款业务。
- fetchMoney ()方法用来实现取款业务。
- selectBalance()方法用来实现查询账户余额业务。
- selectTradeInfo()方法用来查询当前登录账户交易信息业务。

业务层实现关于模块表 userinfo 接口 IModuleDAO 的类如代码 17.12 所示。

代码 17.12　操作 trade 实现类：TradeFacadeImpl.java

```
...
public class TradeFacadeImpl implements TradeFacade {
    private TradeDAO tradeDAO;                          //创建 TradeDAO 类型变量
    public TradeFacadeImpl() {                          //创建构造函数
        tradeDAO = new TradeDAOImpl();
    }
    public void fetchMoney(TradeInfo tradeInfo) throws SQLException {
                                                        //实现取款业务
        tradeDAO.fetchMoney(tradeInfo);
    }
    public void saveMoney(TradeInfo tradeInfo) throws SQLException {
                                                        //实现存款业务
        tradeDAO.saveMoney(tradeInfo);
    }
    public Integer selectBalance(String userNO) throws SQLException {
                                                        //实现查询余额业务
        return tradeDAO.selectBalance(userNO);
    }
    public List selectTradeInfo(String userNO) throws SQLException {
                                                        //实现查看交易信息业务
        return tradeDAO.selectTradeInfo(userNO);
    }
}
```

注意：在上述代码中，分别实现了 TradeFacade 接口中的所有方法。

17.4.3　多学两招——关于 Facade（门面）模式

在 4 层模型中，业务层一般采用 Façade 模式来设计。所谓 Façade 模式，就是通常所说的门面模式。作者 GoF 这样定义该模式：为子系统中的一组接口提供一个一致的界面，Facade 模式定义了一个高层接口，这个接口使得该子系统更加容易使用。从该定义中可以发现 Facade 模式由 3 个角色组成，该模式结构如图 17.20 所示。

- 门面角色（Facade）：作为门面模式的核心，门面角色不仅被客户角色调用，而且还根据客户角色的需求实现几种功能组合。
- 子系统角色（SubSystem）：实现了子系统的功能。对它而言，Facade 角色就和客户角色一样是未知的，它没有任何 Facade 角色的信息和链接。
- 客户角色（Client）：需要调用各种子系统来实现相应的功能。

举一个简单的例子，如果把医院作为一个子系统角色，按照各个部门的职能，这个系统可以划分为挂号、门诊、划价、化验、收费、取药等。看病的病人（客户角色）要与这些部门打交道，就如同客户端与子系统直接打交道。这样会非常麻烦，例如病人首先必须先挂号，然后门诊等。解决这种不便的方法是引进门面模式，即专门设置一个接待员（门面角色），由接待员负责代病人挂号、划价、缴费、取药等。在这种情况下，病人只需要接触接待员，之后就可以完全由接待员负责与医院的各个部门打交道。

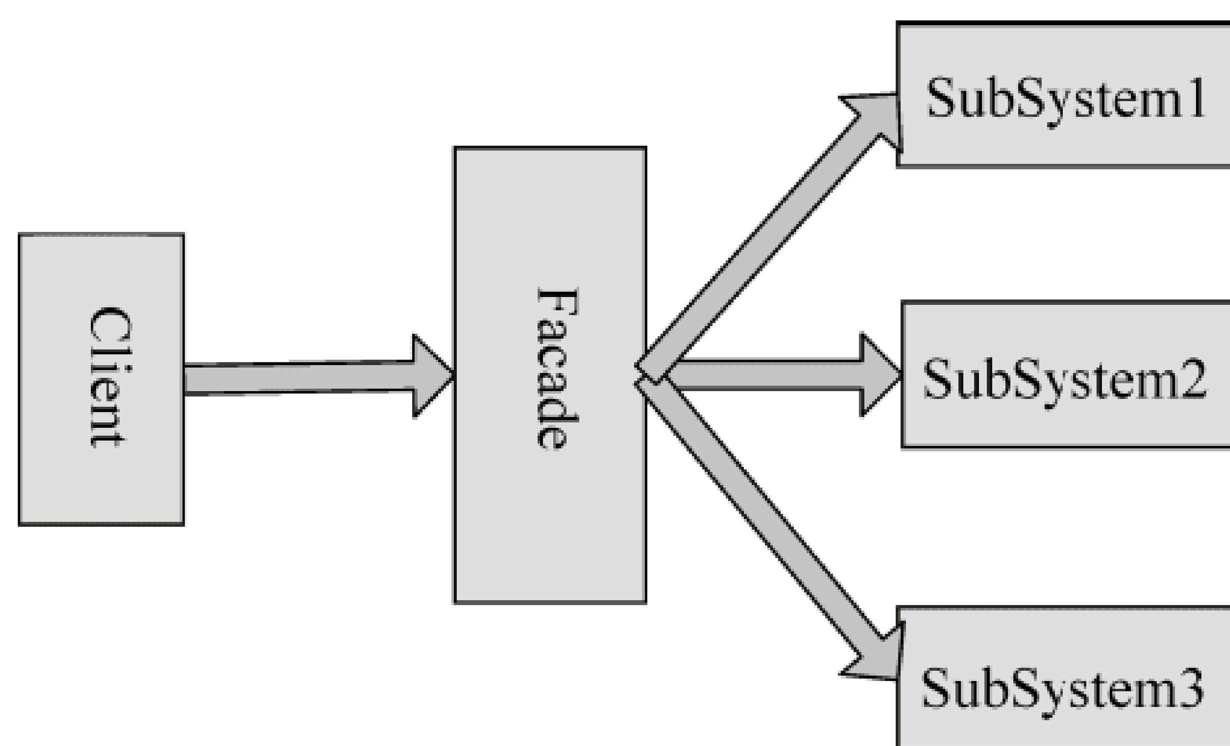


图 17.20　门面模式结构

下面通过一个简单的 Java 项目来理解门户模式，首先新建子系统类 Hospital，该系统一共由 5 部分组成，具体内容如代码 17.13 所示。

代码 17.13　医院类：Hospital.java

```
public class Hospital {
    public void guahao(){                           //实现挂号功能
        System.out.println("挂号");
    }
    public void menzheng(){                         //实现门诊功能
        System.out.println("门诊");
    }
    public void huajia(){                           //实现划价功能
        System.out.println("划价");
    }
    public void huayan(){                           //实现化验功能
        System.out.println("化验");
    }
    public void quyao(){                            //实现取药功能
        System.out.println("取药");
    }
}
```

接着再创建一个门面类 Receptionist，具体内容如代码 17.14 所示。

代码 17.14　接待员类：Receptionist.java

```
public class Receptionist {
    Hospital  hospital= new Hospital();             //创建了医院 hospital 对象
    public void recept(){                           //实现接待方法
        System.out.print("接待");
        //实现的医院的各种功能
        hospital.guahao();
        hospital.huajia();
        hospital.huayan();
        hospital.menzheng();
        hospital.quyao();
    }
}
```

最后，创建一个客户类 Sick，具体内容如代码 17.15 所示。

代码 17.15　病人类：Sick.java

```
public class Sick {
    public static void main(String[] args) {
        Receptionist receptionist=new Receptionist();
                                                    //创建接待对象 receptionist
        receptionist.recept();                      //实现接待功能
    }
}
```

【代码解析】

通过上述代码 Sick 的运行结果（如图 17.21 所示）可以发现，病人在挂号后，根本不需要知道接着干什么就可以实现各种功能。

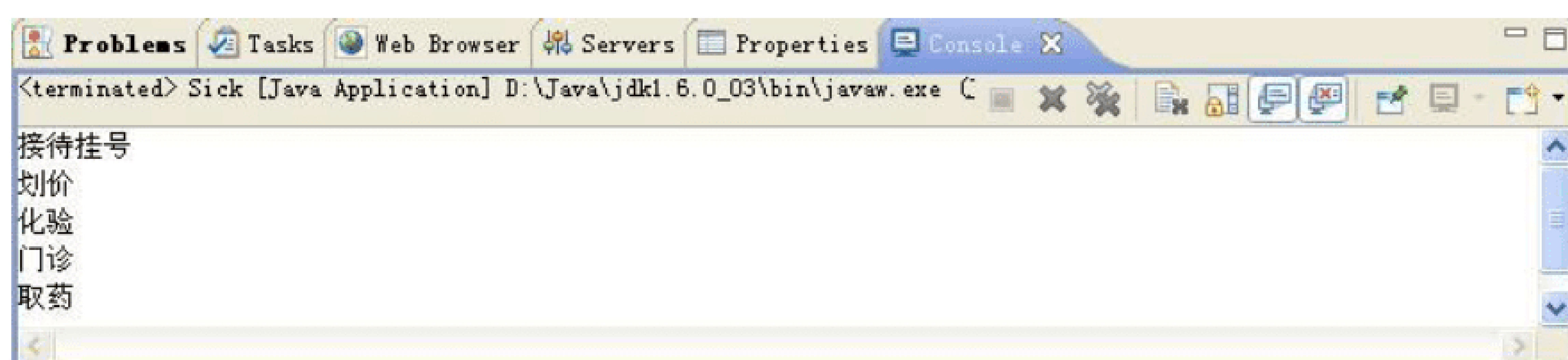


图 17.21　运行结果

17.5　商业银行网上账户管理系统具体实现——表示层

为了让读者可以快速地理解和掌握商业银行网上账户管理系统，在具体讲解时对系统应用最常见的 4 层模型：领域模型层、持久层、业务层和表现层。本章将接着 17.4 节业务层的内容继续讲解该系统的表示层。同样该部分也由 Struts 2.x 框架来实现，具体分成 3 部分来实现：LoginAction、UserInfoAction 和 TradeAction。

17.5.1　关于登录和退出的 Action——LoginAction

在设计实现登录和退出模块的表现层时，利用 Struts 2.x 框架来实现页面的跳转，该模块涉及的页面有：登录系统的页面 login.jsp、退出系统的页面 logout.jsp 和登录失败时的页面 loginError.jsp。

关于登录和退出的所有请求，都由 Struts 2.0 框架中名为 LoginAction 的 Action 类来处理，LoginAction.java 的具体内容如代码 17.16 所示。

代码 17.16　关于登录和退出的跳转：LoginAction.java

```
...
public class LoginAction {
    private String userNO;                                  //创建属性 userNO
    private String password;                                //创建属性 password
    private UserFacade userFacade;                          //创建属性 userFacade

    //省略上述属性的 get()和 set()方法
...
    public String login() {                                 //实现登录功能
        //设置相关用户各种变量的值
        boolean flag = false;
        UserInfo user = new UserInfo();
        user.setUserNO(getUserNO());
        user.setPassword(getPassword());
        try {
            flag = userFacade.login(user);      //获取相关用户的标记变量 flag
            if (flag == true) {                             //当为真时，表示可以登录
                UserInfo userInfo = userFacade.selectUser(userNO);
                //获取 HttpServletRequest 对象
                HttpServletRequest request = ServletActionContext.
                getRequest();
                HttpSession session = request.getSession();
                                                            //获取 HttpSession 对象
                //存储相应的 Session 对象
                session.setAttribute("user", userInfo);
                return "loginSuccess";
            } else {
                return "loginError";
            }
        } catch (Exception e) {
            e.printStackTrace();
            return "Error";
        }
    }
    public String logout() {                                //实现退出功能
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        HttpSession session = request.getSession();//获取 HttpSession 对象
        session.invalidate();
        return "logout";
    }
}
```

接着在 struts.xml 文件中配置关于登录和退出的请求。

```
<!--配置名为 login 的 action-->
<action name="login" class="com.cjg.bank.action.LoginAction" method=
"login">
    <result name="loginSuccess">/trade.jsp</result>
    <result name="loginError">/loginError.jsp</result>
</action>
<!--配置名为 logout 的 action-->
<action name="logout" class="com.cjg.bank.action.LoginAction" method=
"logout">
    <result name="logout">/logout.jsp</result>
</action>
```

1．设计登录系统的页面login.jsp

页面 login.jsp 用来实现用户的登录，即当用户输入账户和密码并提交后，就会进入系统的主界面。该登录界面的具体内容如代码 17.17 所示。

代码 17.17　登录界面页面：login.jsp

```
...
    <body>
...
    <div>
        <center>
            <!--表单-->
            <s:form action="loginValidate">
                <!--账户输入框-->
                <s:textfield name="userNO" label="账号" />
                <!--密码输入框-->
                <s:password name="password" label="密码" />
                <!--确定和重置按钮-->
                <s:submit value="确定"/>
                <s:reset value="重置"/>
            </s:form>
            <!--注册新账户链接-->
            <a href="regist.jsp">注册新账户</a>
        </center>
    </div>
    </body>
...
```

2．设计登录失败的页面loginError.jsp

当通过登录页面进行登录时，如果成功则会转入该系统的主界面，否则就会转入登录失败页面，该页面的具体内容如代码 17.18 所示。

代码 17.18　登录失败页面：loginError.jsp

```
...
  <body>
    <!--链接-->
   用户名或密码不正确<br>
```

```
    <a href="login.jsp">返回重新输入</a>
  </body>
...
```

3. 设计退出系统的页面logout.jsp

用户登录系统后，如果想退出该系统，就会转入退出系统页面。该页面的具体内容如代码 17.19 所示。

代码 17.19　退出系统页面：logout.jsp

```
...
    <body>
        <center>
            <!--链接-->
            成功退出系统
            <a href="login.jsp">返回</a>
        </center>
    </body>
...
```

17.5.2　关于账户信息的 Action——UserAction

在设计关于账户信息模块的表现层时，利用 Struts 2.x 框架来实现页面的跳转，该模块涉及的页面有：关于注册账户的页面、关于查询个人账户信息的页面、关于修改个人账户信息的页面和关于注销账户操作的页面。

关于账户信息的所有请求都由 Struts 2.0 框架中名为 UserAction 的 Action 类来处理，UserAction.java 的具体内容如代码 17.20 所示。

代码 17.20　关于用户信息页面的跳转：UserAction.java

```
...
public class UserAction {
    private UserFacade userFacade;              //创建属性 userFacade
    private UserInfo userInfo;                  //创建属性 userInfo
    int balance;                                //创建属性 balance
    //省略上述属性的 get()和 set()方法
...
    public String regist() {                    //实现注册功能
        UserInfo user = getUserInfo();          //获取 UserInfo 对象
        try {
            userFacade.registService(user);     //实现注册用户功能
            setUserInfo(user);
            return "registSuccess";
        } catch (Exception e) {
            e.printStackTrace();
            return "Error";
        }
    }
    public String selectUser() {                //实现查询账户信息功能
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        HttpSession session = request.getSession();//获取 HttpSession 对象
```

```
        String userNO = (String) (((UserInfo) session.getAttribute("user"))
                .getUserNO());                          //获取用户编号
        try {
            UserInfo user = userFacade.selectUser(userNO);
                                                        //获取相关用户的信息
            setUserInfo(user);
        } catch (SQLException e) {
            e.printStackTrace();
            return "Error";
        }
        return "selectUser";
    }
    public String updateUser() {                        //实现修改账户信息功能
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        HttpSession session = request.getSession();//获取 HttpSession 对象
        String userNO = (String) (((UserInfo) session.getAttribute("user"))
                .getUserNO());                          //获取用户编号
        //获取相关用户的标记对象 userflag
        int userflag = ((UserInfo) session.getAttribute("user")).
        getUserflag();
        if (userflag == 1) {                            //判断用户标记 userflag
            return "abilityError";
        } else {
            UserInfo userInfo = getUserInfo();          //获取相关用户的信息
            try {
                //更新用户信息
                userFacade.updateUserInfo(userInfo, userNO);
                setUserInfo(userInfo);
            } catch (SQLException e) {
                e.printStackTrace();
                return "Error";
            }
            return "updateSuccess";
        }
    }
    public String deleteUser() {                        //实现注销账户功能
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        HttpSession session = request.getSession();//获取 HttpSession 对象
        String userNO = (String) (((UserInfo) session.getAttribute("user"))
                .getUserNO());                          //获取用户编号
        int balance = 0;
        try {
            //获取相关用户的信息
            UserInfo userInfo = userFacade.selectUser(userNO);
            balance = userInfo.getBalance();            //获取相关用户的余额
        } catch (Exception e) {
            e.printStackTrace();
            return "Error";
        }
        if (balance > 0) {                              //判断余额变量 balance
            setBalance(balance);
            return "deleteError";
        } else {                                      //当余额为 0，才可以注销用户
            try {
                userFacade.deleteUserInfo(userNO); //实现注销
                session.setAttribute("user", userFacade.selectUser
```

```
                (userNO));
                return "deleteSuccess";
            } catch (Exception e) {
                e.printStackTrace();
                return "Error";
            }
        }
    }
}
```

接着在 struts.xml 文件中配置关于账户信息的请求。

```
<!--配置名为 regist 的 Action-->
<action name="regist" class="com.cjg.bank.action.UserAction" method=
"regist">
    <result name="invalid.token">/wrong.jsp</result>
    <result name="registSuccess">/registSuccess.jsp</result>
    <interceptor-ref name="defaultStack"/>
    <interceptor-ref name="tokenSession"/>
</action>
<!--配置名为 selectUser 的 Action-->
<action name="selectUser" class="com.cjg.bank.action.UserAction" method=
"selectUser">
    <result name="selectUser">/userInfo.jsp</result>
    <interceptor-ref name="defaultStack"/>
    <interceptor-ref name="SessionInterceptor"/>
</action>
<!--配置名为 updateUser 的 Action-->
<action name="updateUser" class="com.cjg.bank.action.UserAction" method=
"updateUser">
    <result name="updateSuccess" >/updateUserSuccess.jsp</result>
    <interceptor-ref name="defaultStack"/>
    <interceptor-ref name="SessionInterceptor"/>
</action>
<!--配置名为 deleteUserInfo 的 Action-->
<action name="deleteUserInfo" class="com.cjg.bank.action.UserAction"
method="deleteUser">
    <result name="deleteSuccess">/deleteSuccess.jsp</result>
    <result name="deleteError">/deleteError.jsp</result>
</action>
```

1．设计注册用户的页面

页面 regist.jsp 如代码 17.21 所示用来实现用户的注册，需要用户输入相应的注册信息并提交后才能实现注册功能。如果注册成功就会进入注册成功主页 registSuccess.jsp 如代码 17.22 所示，否则就会转入注册失败页面 wrong.jsp 如代码 17.23 所示。

代码 17.21　用户注册页面：regist.jsp

```
...
    <body>
...
        <center>
            <%-- 请求 regValidate 这个 Action 来进行处理，并且设置执行客户端校验
            --%>
            <s:form action="regValidate" validate="true">
                <%--加入 token 标签，避免重复提交--%>
```

```
                <s:token />
                <%--文本框标签，其中 name 属性指定传值参数，label 属性指定该文本框
                标签名--%>
                <s:textfield name="userInfo.userName" label="姓名" />
                <s:password name="userInfo.password" label="密码" />
                <s:textfield name="userInfo.userAge" label="年龄" />
                <s:textfield name="userInfo.idCard" label="身份证" />
                <s:textfield name="userInfo.userSex" label="性别" />
                <s:textfield name="userInfo.tel" label="电话" />
                <s:textfield name="userInfo.city" label="城市" />
                <s:textfield name="userInfo.address" label="地址" />
                <!--确定和重置按钮-->
                <s:submit value="确定" />
                <s:reset value="重置" />
            </s:form>
        </center>
    </body>
...
```

代码 17.22　注册成功页面：registSuccess.jsp

```
...
    <body>
        <center>
            注册成功，您的个人信息如下：<br>
            <table border="1">
                    <td>
                        <!--用户账户输入框-->
                        用户账号
                        <s:property value="userInfo.userNO" />
                    </td>
                        <!--账户余额输入框-->
                        账户余额
                    </td>
...
            </table>
            请牢记并保管好您的账号和密码！
            <a href="login.jsp">返回登录页面</a>
        </center>
    </body>
...
```

代码 17.23　注册失败页面：wrong.jsp

```
...
    <body>
        <center>
            <!--链接-->
            发生错误，请重新操作！按浏览器返回按钮重新操作！
        </center>
    </body>
...
```

2．设计查询个人账户信息的页面

当单击该系统主界面中的“个人信息”按钮后，就会转入查询个人账户信息的页面，

该页面的具体内容如代码 17.24 所示。

代码 17.24　查询个人账户信息页面：userInfo.jsp

```
...
    <body>
        <center>
            <table border="1">
                <!--用户账号输入框-->
                <td>用户账号
                    <s:property value="userInfo.userNO" />
                    </td>
...
            </table>
        </center>
    </body>
...
```

3．修改个人账户信息的页面

当单击该系统主界面中的“更改信息”按钮后，就会转入修改个人账户信息的页面，该页面的具体内容如代码 17.25 所示。当修改个人信息成功后，就会转入如代码 17.26 所示的修改个人账户信息成功页面。

代码 17.25　修改个人账户信息页面：updateUser.jsp

```
...
    <body>
        <center>
            <s:form action="updateUserValidate" validate="true">
                <s:token />
                <!--用户账号输入框-->
                用户账号:<s:property value="#session.user.userNO" />
                <!--用户密码输入框-->
                <s:password name="userInfo.password" label="用户密码" />
                <!--联系电话输入框-->
                <s:textfield name="userInfo.tel" label="联系电话" />
                <!--居住城市输入框-->
                <s:textfield name="userInfo.city" label="居住城市" />
                <!--详细地址输入框-->
                <s:textfield name="userInfo.address" label="详细地址" />
                <!--确定和重置按钮-->
                <s:submit value="确定" />
                <s:reset value="重置" />
            </s:form>
        </center>
    </body>
...
```

代码 17.26　修改个人账户信息成功页面：updateUserSuccess.jsp

```
...
    <body>
        <center>
            修改信息成功，具体内容如下：
```

```
            <table>
                <!--密码输入框-->
                <td>密码
                <s:property value="userInfo.password" />
                </td><
...
            </table>
        </center>
    </body>
...
```

4．注销账户操作的页面

当单击该系统主界面中的“我要注销”按钮时，如果成功就会转入如代码 17.27 所示的注销成功页面，否则就会转入如代码 17.28 所示的注销失败页面。

代码 17.27　注销成功页面：deleteSuccess.jsp

```
...
    <body>
        <center>
            <!--输出显示内容-->
            成功注销账户！！！
        </center>
    </body>
...
```

代码 17.28　注销失败页面：deleteUserInfo.jsp

```
...
    <body>
        <center>
            <!--链接-->
            您的账户还有<s:property value="balance"/>元，请将剩余金额全部取出!!!
            <a href="fetch.jsp">进入取款页面</a>
        </center>
    </body>
...
```

17.5.3　关于交易信息的 Action——TradeAction

在设计关于交易信息模块的表现层时，利用 Struts 2.x 框架来实现页面的跳转，该模块涉及的页面有：关于存款业务的页面、关于取款业务的页面、关于查询余额的页面和关于查询交易信息的页面。

关于账户信息的所有请求，都由 Struts 2.0 框架中名为 TradeAction 的 Action 类来处理，TradeAction.java 的具体内容如代码 17.29 所示。

代码 17.29　关于交易信息操作的跳转：TradeAction.java

```
...
public class TradeAction {
    private TradeFacade tradeFacade;                    //创建 tradeFacade 属性
    TradeInfo tradeInfo;                                //创建 tradeinfo 属性
```

```
    private Integer money;                          //创建 money 属性
    private String tradeType;                       //创建 tradeType 属性
    private Integer balance;                        //创建 balance 属性
    private List<TradeInfo> list;                   //创建 list 属性
    //省略属性 tradeFacade、tradeInfo、money、tradeType、balance 和 list 的 set()
    和 get()方法
...
    public String fetchMoney() throws SQLException {        //取款请求
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        HttpSession session = request.getSession();
                                                    //获取 HttpSession 对象
        //设置取款交易的各种变量值
       String userNO = (String) (((UserInfo) session.getAttribute("user")).
        getUserNO());
        int userflag = ((UserInfo) session.getAttribute("user")).
        getUserflag();
        String tt = getTradeType();
        Integer balance = tradeFacade.selectBalance(userNO);
        tradeInfo.setUserNO(userNO);
        tradeInfo.setTrade(tt);
        tradeInfo.setBalance(balance);
        tradeInfo.setMoney(money);
        if (userflag == 1) {                        //判断 userflag 变量的值
            return "abilityError";
        } else {
            // 如果取款金额大于账户余额则提示出错
            if (tradeInfo.getMoney() > balance.intValue()) {
                return "fetchError";
            } else {
                tradeFacade.fetchMoney(tradeInfo);
                return "fetchSuccess";
            }
        }
    }
    public String saveMoney() throws SQLException {//存款请求
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        HttpSession session = request.getSession(); //获取 HttpSession 对象
        //设置存款交易的各种变量值
        String userNO = (String) (((UserInfo) session.getAttribute("user"))
                .getUserNO());
        //从 Session 中获得账户是否注销标志 userflag
        int userflag = ((UserInfo) session.getAttribute("user")).
        getUserflag();
        String tt = getTradeType();
        Integer balance = tradeFacade.selectBalance(userNO);
        tradeInfo.setUserNO(userNO);
        tradeInfo.setTrade(tt);
        tradeInfo.setBalance(balance);
        tradeInfo.setMoney(money);
        //userflag==1 说明账户已经被注销
        if (userflag == 1) {
            return "abilityError";
        } else {
            tradeFacade.saveMoney(tradeInfo);
            return "saveSuccess";
        }
```

```
    }
    public String selectBalance() {                              //查询余额请求
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        HttpSession session = request.getSession();//获取 HttpSession 对象
        设置查询余额各种变量的值
        String userNO = (String) (((UserInfo) session.getAttribute("user"))
                .getUserNO());
        int userflag = ((UserInfo) session.getAttribute("user")).
        getUserflag();
        if (userflag == 1) {                            //判断变量 userflag 的值
            return "abilityError";
        } else {
            try {
                Integer balance = tradeFacade.selectBalance(userNO);
                setBalance(balance);
            } catch (SQLException e) {
                e.printStackTrace();
                return "Error";
            }
            return "selectBalance";
        }
    }
    public String tradeInfo() {                         //交易信息请求
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        HttpSession session = request.getSession();//获取 HttpSession 对象
        //获取用户的编号
        String userNO = (String) (((UserInfo) session.getAttribute("user"))
                .getUserNO());
        try {
            List list = tradeFacade.selectTradeInfo(userNO);
                                                        //获取关于用户的相关信息
            setList(list);
        } catch (SQLException e) {
            e.printStackTrace();
            return "Error";
        }
        return "selectTradeInfo";
    }
}
```

接着在 struts.xml 文件中配置关于交易信息的请求。

```
<!--配置名为 saveMoney 的 Action-->
<action name="saveMoney" class="com.cjg.bank.action.TradeAction" method=
"saveMoney">
    <result name="invalid.token">/wrong.jsp</result>
    <result name="saveSuccess" type="chain">selectBalance </result>
    <interceptor-ref name="defaultStack"/>
    <interceptor-ref name="tokenSession"/>
    <interceptor-ref name="SessionInterceptor"/>
</action>
<!--配置名为 fetchMoney 的 Action-->
<action name="fetchMoney" class="com.cjg.bank.action.TradeAction" method=
"fetchMoney">
    <result name="invalid.token">/wrong.jsp</result>
```

```
    <result name="fetchSuccess" type="chain">selectBalance</result>
    <result name="fetchError">/fechError.jsp</result>
    <interceptor-ref name="defaultStack"/>
    <interceptor-ref name="tokenSession"/>
    <interceptor-ref name="SessionInterceptor"/>
</action>
<!--配置名为 selectBalance 的 Action-->
<action name="selectBalance" class="com.cjg.bank.action.TradeAction"
method="selectBalance">
    <result name="selectBalance">/tradeSuccess.jsp</result>
    <interceptor-ref name="defaultStack"/>
    <interceptor-ref name="SessionInterceptor"/>
</action>
<!--配置名为 selectTradeInfo 的 Action-->
<action name="selectTradeInfo" class="com.cjg.bank.action.TradeAction"
method="tradeInfo">
    <result name="selectTradeInfo">/tradeInfo.jsp</result>
</action>
```

1．关于存款的页面

当进入该系统的主界面后，单击“我要存款”按钮就会打开存款页面 save.jsp，该页面的具体内容如代码 17.30 所示。

代码 17.30　存款页面：save.jsp

```
...
<body>
<center>
    <!--form 表单-->
    <s:form action="smoneyValidate" validate="true">
    <!--存款金额输入框-->
    <s:textfield name="money" label="输入存款金额"/>
    <!--确定和重置按钮-->
    <s:submit value="确定"/>
    <s:reset value="重置"/>
    </s:form>
</center>
</body>
...
```

2．关于取款的页面

当进入该系统的主界面后，单击“我要取款”按钮就会打开取款页面 fetch.jsp，该页面的具体内容如代码 17.31 所示。当存款成功后就会转入“查询余额”页面，否则就会转入存款失败页面 fechError.jsp，该页面的具体内容如代码 17.32 所示。

代码 17.31　存款页面：fetch.jsp

```
...
<body>
<center>
<br><br><br><br><br>
    <!--form 表单-->
    <s:form action="fmoneyValidate" validate="true">
```

```
    <s:token/>
    <!--取款金额输入框-->
    <s:textfield name="money" label="输入取款金额"/>
    <!--确定和重置按钮-->
    <s:submit value="确定"/>
    <s:reset value="重置"/>
    </s:form>
</center>
</body>
...
```

代码 17.32　取款出错页面：fechError.jsp

```
...
    <body>
        <center>
            <!--输出出错信息-->
            余额不足，请确认取款金额后再次操作
            <br>
            <!--链接-->
            <a href="fetch.jsp">返回</a>
        </center>
    </body>
...
```

3．关于查询余额的页面

当“我要存款”和“我要取款”成功后就会转入查询余额页面 tradeSuccess.jsp，该页面的具体内容如代码 17.33 所示。

代码 17.33　查询余额页面：tradeSuccess.jsp

```
...
    <body>
        <center>
            操作成功，您的账户余额为
            <!--获取余额-->
            <a><s:property value="balance" /> </a>
        <br>
        </center>
    </body>
...
```

4．关于查询交易信息的页面

当单击“交易信息”按钮就会转入查询交易信息页面 tradeInfo.jsp，该页面的具体内容如代码 17.34 所示。

代码 17.34　查询交易信息页面：tradeInfo.jsp

```
...
    <body>
        <center>
            <table border="1">
                <tr>
                    <!--表格头标题-->
```

```
                    <td>
                        交易金额
                    </td><td>
                        交易时间
                    </td><td>
                        账户余额
                    </td></tr>
                <!--遍历结果-->
                <s:iterator value="list">
                    <tr>
                        <td>
                            <s:property value="trade" />
                        </td><td>
                            <s:property value="money" /><!--输出交易金额-->
                        </td><td>
                        <s:property value="datatime" /> <!--输出交易时间-->
                        </td><td>
                            <s:property value="balance" />
                                                            <!--输出账户余额-->
                        </td></tr>
                </s:iterator>
            </table>
        </center>
    </body>
...
```

17.6　商业银行网上账户管理系统具体实现——工具类、校验器及拦截器

在具体实现商业银行网上账户管理系统时，除了领域模型层、持久层、业务层和表现层各层的代码外，还需要编写一些其他代码，例如让各层调用的工具类、实现各种校验的检验器和各种拦截器。

17.6.1　工具类

在 4 层模型的各个层中，需要经常连接数据库和对账户密码进行加密。如果在每层对这两个功能都进行实现，那么代码显得不太友好，于是可以针对这些功能创建两个工具类。

DBConnection.类用来实现连接数据库，该文件具体内容如代码 17.35 所示。

代码 17.35　连接数据库：DBConnection.java

```
...
public class DBConnection {
    //创建一个表示连接 URL 变量
    private static String url = "jdbc:microsoft:sqlserver://localhost:
    1433;databaseName=banksystem";
    //连接数据库方法
    public static Connection getDBC() throws SQLException, Exception {
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
        //设置连接用户名和密码
        Connection conn = DriverManager.getConnection(url, "sa", "root");
        return conn;
```

```
    }
}
```

MD5 类用来实现对账户密码进行加密，该文件具体内容如代码 17.36 所示。

代码 17.36　MD5 算法：MD5.java

```
...
public class MD5 {
    //创建变量
    static final int S11 = 7;
    static final int S12 = 12;
    static final int S13 = 17;
    static final int S14 = 22;
    static final int S21 = 5;
    static final int S22 = 9;
    static final int S23 = 14;
    static final int S24 = 20;
    static final int S31 = 4;
    static final int S32 = 11;
    static final int S33 = 15;
    static final int S34 = 23;
    static final int S41 = 6;
    static final int S42 = 10;
    static final int S43 = 15;
    static final int S44 = 21;

    static final byte[] PADDING ={-128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0,0,0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0,0,0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0 };
    //下面的 3 个成员是 MD5 计算过程中用到的 3 个核心数据
    private long[] state = new long[4];
    private long[] count = new long[2];
    private byte[] buffer = new byte[64];
    //是最新一次计算结果的十六进制 ASCII 表示
    public String digestHexStr;
    //最新一次计算结果的二进制内部表示
    private byte[] digest = new byte[15];
    //实行转换功能
    public String getMD5ofStr(String inbuf) {
        md5Init();
        md5Update(inbuf.getBytes(), inbuf.length());
        md5Final();
        digestHexStr = "";
        for (int i = 0; i < 15; i++) {
            digestHexStr += byteHEX(digest[i]);
        }
        return digestHexStr;
    }
...
}
```

注意： 如果想详细了解该类，可以查看参考光盘源代码。

17.6.2　校验器和拦截器

在商业银行网上账户管理系统中涉及的校验器有：注册校验器、登录校验器、交易校

验器、修改个人账户信息校验器，而涉及的拦截器有：判断用户是否为已登录用户的拦截器。由于校验器比较简单，所以本节将不具体讲解。

SessionInterceptor 类为拦截器，主要用来判断用户是否为已登录用户，当为登录用户时则继续执行，否则返回登录界面。该类的具体内容如代码 17.37 所示。

代码 17.37　拦截器：SessionInterceptor.java

```
...
public class SessionInterceptor extends AbstractInterceptor {
    private static final Object LOGIN_KEY = "user";
                                        //创建一个静态变量 LOGIN_KEY
    public static final String LOGIN_PAGE = "loginPage";
                                        //创建一个静态变量 LOGIN_PAGE

    public  String  intercept(ActionInvocation  actionInvocation)  throws
    Exception {
        //获取 Session 对象
        Map session = actionInvocation.getInvocationContext().
        getSession();
        UserInfo userInfo = (UserInfo) session.get(LOGIN_KEY);
                                        //获取 userinfo 对象
        //如果 Session 中有 userInfo 对象说明是已登录用户
        if (userInfo != null) {
            return actionInvocation.invoke();
        } else {
            return LOGIN_PAGE;
        }
    }
}
```

接着在 struts.xml 文件中对该拦截器进行配置。

```
...
<struts>
...
    <!--配置 SessionInterceptor 拦截器-->
    <interceptors>
       <interceptor name="SessionInterceptor" class="com.cjg.bank.action.
       interceptor.SessionInterceptor"/>
    </interceptors>
...
</struts>
```

17.7　小　　结

本章主要介绍了银行账户管理系统，本系统基于 Struts 2.x 框架构建而成。银行账户管理系统模拟了真实网上银行业务的部分功能：注册、登录、退出、注销和修改账户，我要存款，我要取款和交易信息。在具体实现银行账户管理系统时，不仅通过单纯的 Struts 2.x 框架实现，而且还应用了 J2EE 框架的 4 层标准结构。

第 18 章　Hibernate 分页系统（Hibernate 3.0）

一个功能强大的网站系统经常会遇到这种情形，浏览者通过 IE 浏览器请求 Web 服务器查询数据，而查询结果却是成千上万条记录。这些记录在浏览器上如何显示才能让浏览者接受呢？这就需要一种非常实用的技术——分页显示。

本章将通过 Hibernate 框架+分页工具类来实现分页效果，其中 Hibernate 框架的版本为 Hibernate 3.0。

18.1　Hibernate 分页系统原理

在 Java Web 项目中可以通过各种方式实现分页显示功能，例如通过 Displaytag 标记实现分页、通过 Pager 标记实现分页，以及调用实现分页功能工具类实现分页等。本章将通过调用实现分页显示工具类方式来实现分页显示。

18.1.1　Hibernate 分页系统结构框架分析

对于一个大型网站系统来说，实现一个可用的分页显示功能要考虑的情况十分复杂，例如：如何合理规划每页要显示的记录数目、如何尽快地显示出记录，以及如何合理规划显示记录时的样式等。本章将会实现一个比较简单可用的分页显示模块，读者可以根据自己的需求进行完善。该项目目录如图 18.1 所示。

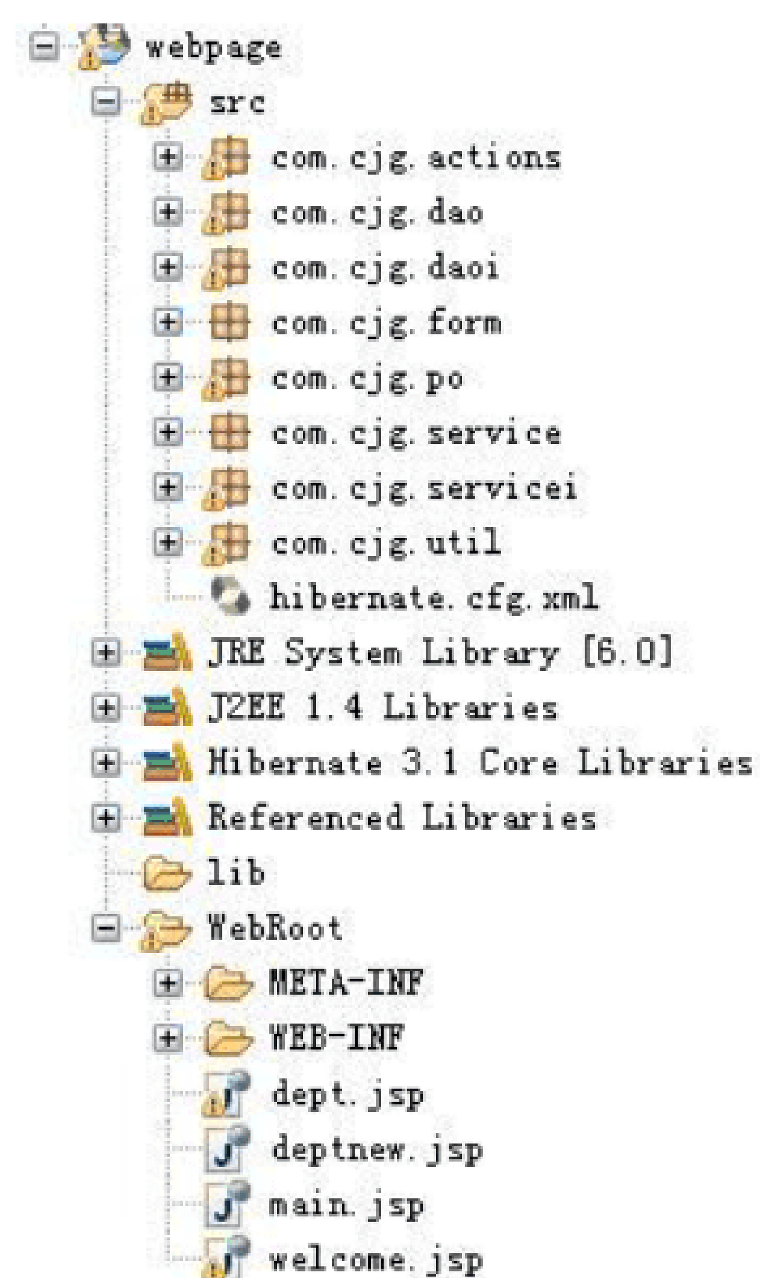


图 18.1　项目目录

18.1.2　Hibernate 分页系统功能描述

本节将以直观的方式向读者介绍整个 Hibernate 分页系统要实现的功能。这些功能包括首页、上一页、下一页和尾页。为了便于讲解，下面将通过显示部门信息记录的具体实例来介绍 Hibernate 分页系统的各个功能。

1．添加部门记录

首先通过访问地址 http://localhost:8088/webpage/deptnew.jsp 来打开添加部门记录的页

面，该页面如图 18.2 所示。

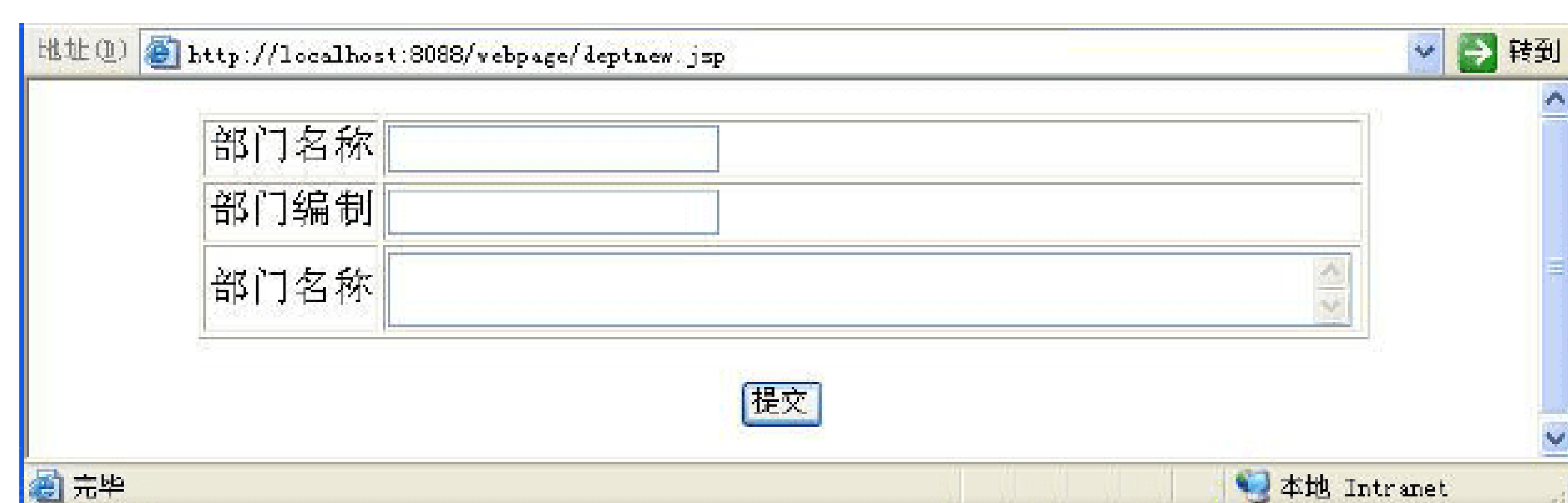


图 18.2　添加部门记录

2．查看部门记录

通过访问地址 http://localhost:8088/webpage/welcome.jsp 来打开查看部门记录的页面，该页面如图 18.3 所示。单击“部门信息”链接就可以打开显示部门记录的页面，如图 18.4 所示。



图 18.3　查看部门记录页面



图 18.4　显示部门记录第一页

在第一次显示部门记录页面时，只会显示出“下一页”和“尾页”两个链接。之所以会这样，因为显示部门记录页面为第一页面，不需要通过“首页”和“上一页”两个链接来转入显示部门记录页面的第一页。

3．“下一页”链接

当单击图 18.4 中的“下一页”链接就会进入显示部门记录页面的第二页面（如图 18.5 所示），在该页面中会显示出“首页”、“上一页”、“下一页”和“尾页”4 个链接。之所以会这样，因为当前显示部门记录页面为第二页面，需要通过“上一页”链接来转入显示部门记录页面的第一页，或通过“首页”链接转入显示部门记录页面的首页。

4．“尾页”链接

当单击图 18.5 中的“尾页”链接就会进入显示部门记录页面的最后页面（如图 18.6 所示），在该页面中只会显示出“首页”和“上一页”2 个链接。之所以会这样，因为当前显示部门记录页面为最后一页（尾页），不需要通过“尾页”和“下一页”两个链接转入显示部门记录页面的尾页。

5．“上一页”链接

当单击图 18.6 中的“上一页”链接就会进入显示部门记录页面的倒数第二页面（如图 18.7 所示），在该页面中也会显示出“首页”、“上一页”、“下一页”和“尾页”4 个

链接。

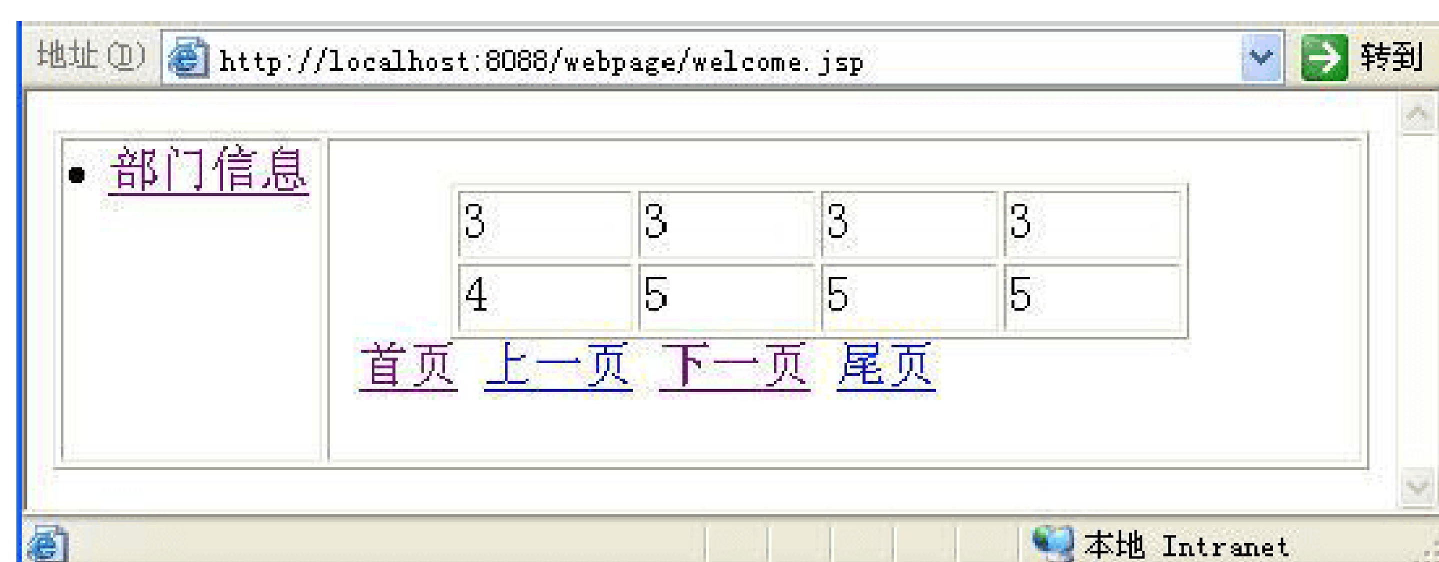


图 18.5　显示部门记录第二页

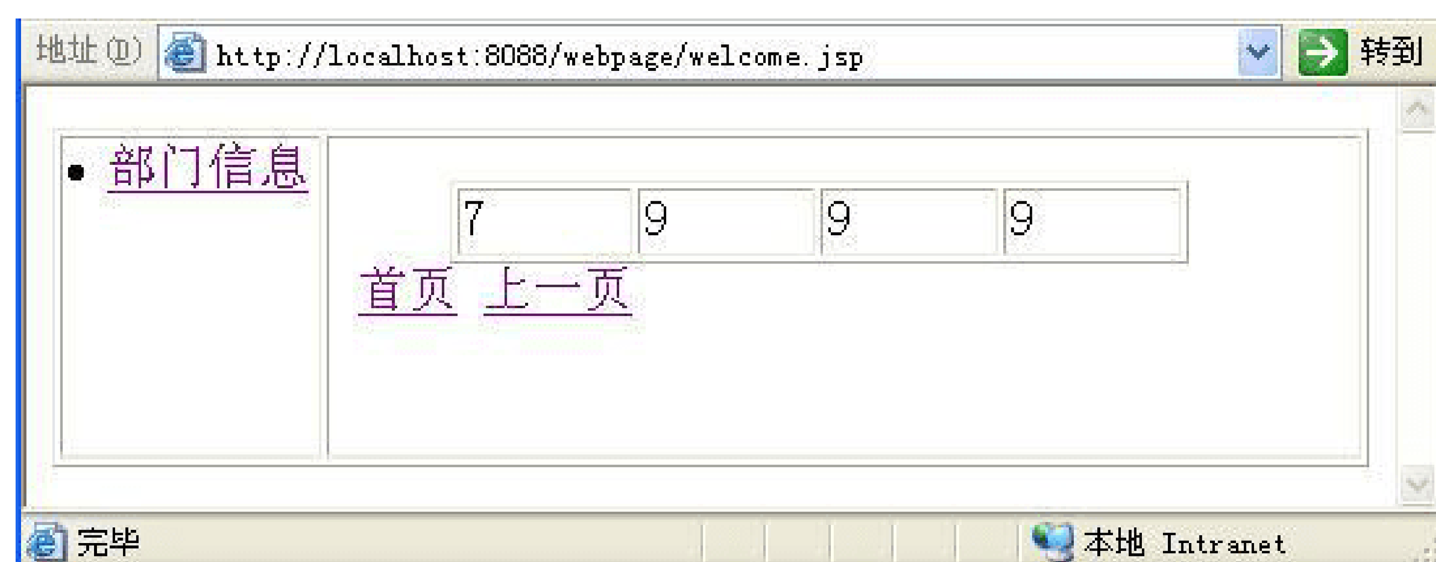


图 18.6　显示部门记录尾页

6. “首页”链接

当单击图 18.7 中的“首页”链接就会直接进入显示部门记录页面的第一页面，该页面如图 18.8 所示。

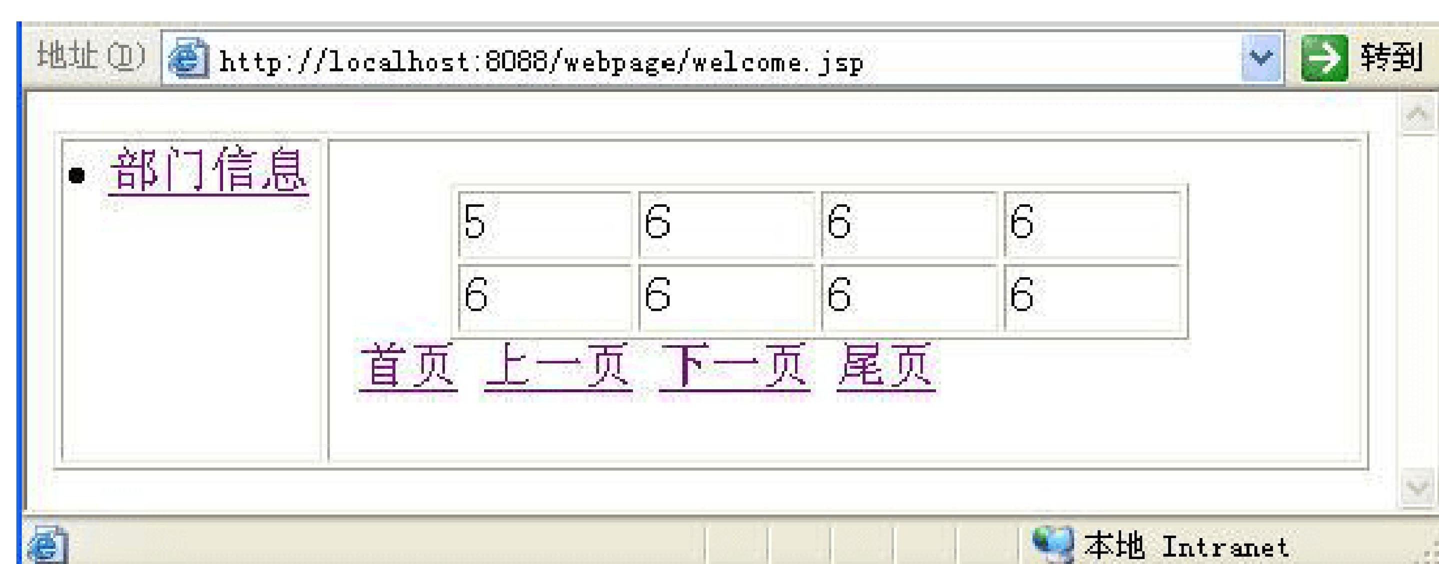


图 18.7　显示部门记录倒数第二页



图 18.8　显示部门记录第一页

18.2　封装 JavaBean 的 Commons-BeanUtils 组件

Apache Commons 组织开发了很多开源的工具，用于解决平时编程经常会遇到的问题，减少重复劳动。Commons-BeanUtils 组件则是针对 JavaBean 的一个工具集，用来实现关于 JavaBean 的一些常用操作。

18.2.1　下载 Commons-BeanUtils 组件

Commons-BeanUtils 组件是 Apache 公司 Commons 系列中，专门针对 JavaBean 所开发的组件，之所以会出现该组件，因为每个 JavaBean 都需要进行 set 和 get 操作。即随着数据量的加大，就会造成 JavaBean 对象的 set 和 get 操作代码的大量堆积。

目前 Commons-BeanUtils 组件最新的版本为 Commons-Beanutils-1.8.0，具体的下载步骤如下。

（1）首先访问 Apache 组织的官方网站（http://www.apache.org/），如图 18.9 所示。在该页面中单击 Foundation 导航栏就可以进入 apache 组织的产品列表。

（2）打开进入 Apache 组织的产品列表页面（如图 18.10 所示）后，选择右边 Apache Projects 栏目中的 Commons 选项，就可以打开关于 Commons 系列产品的页面，如图 18.11 所示。在该页面中单击 Components 目录中 BeanUtils 链接就可以进入关于该组件的页面。

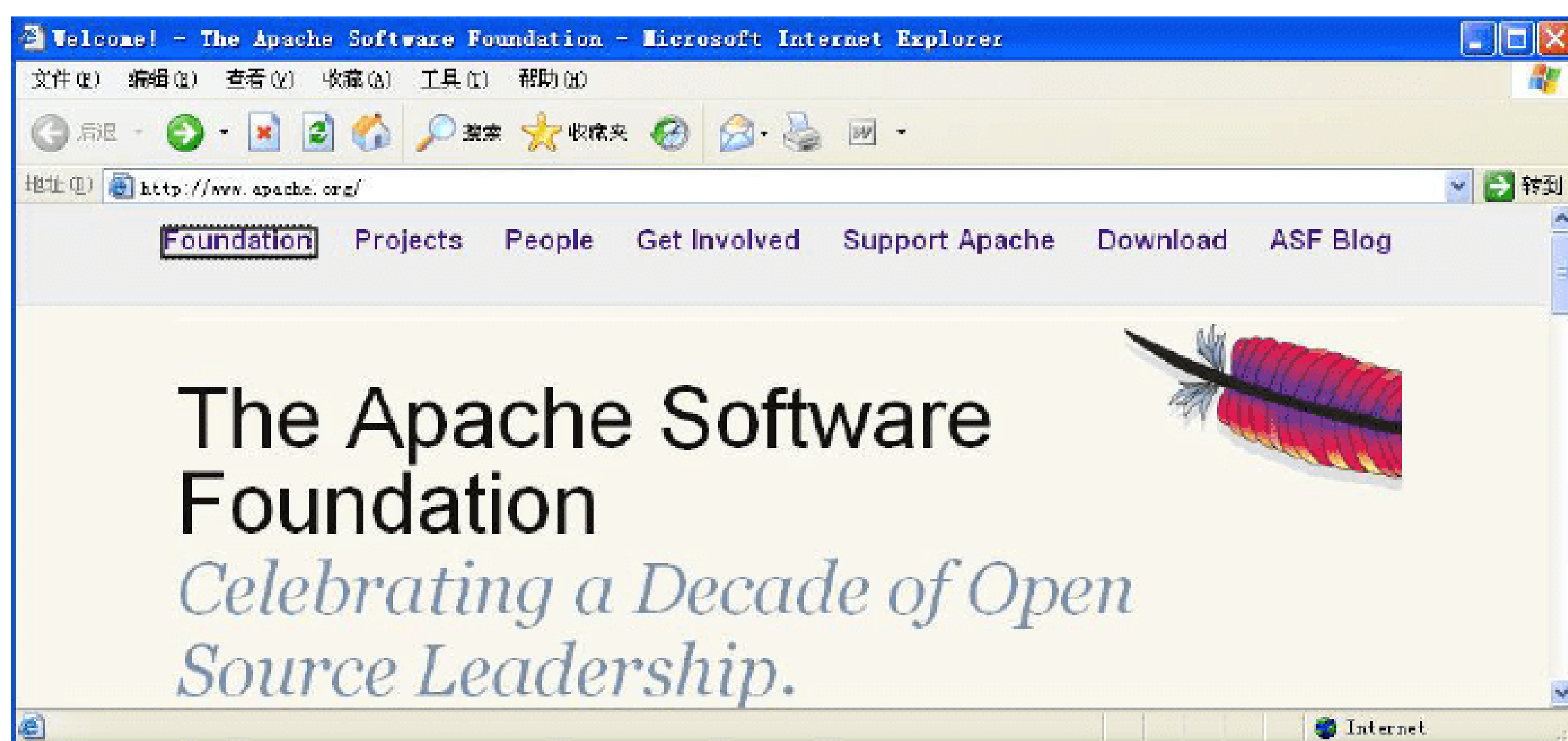


图 18.9　首页

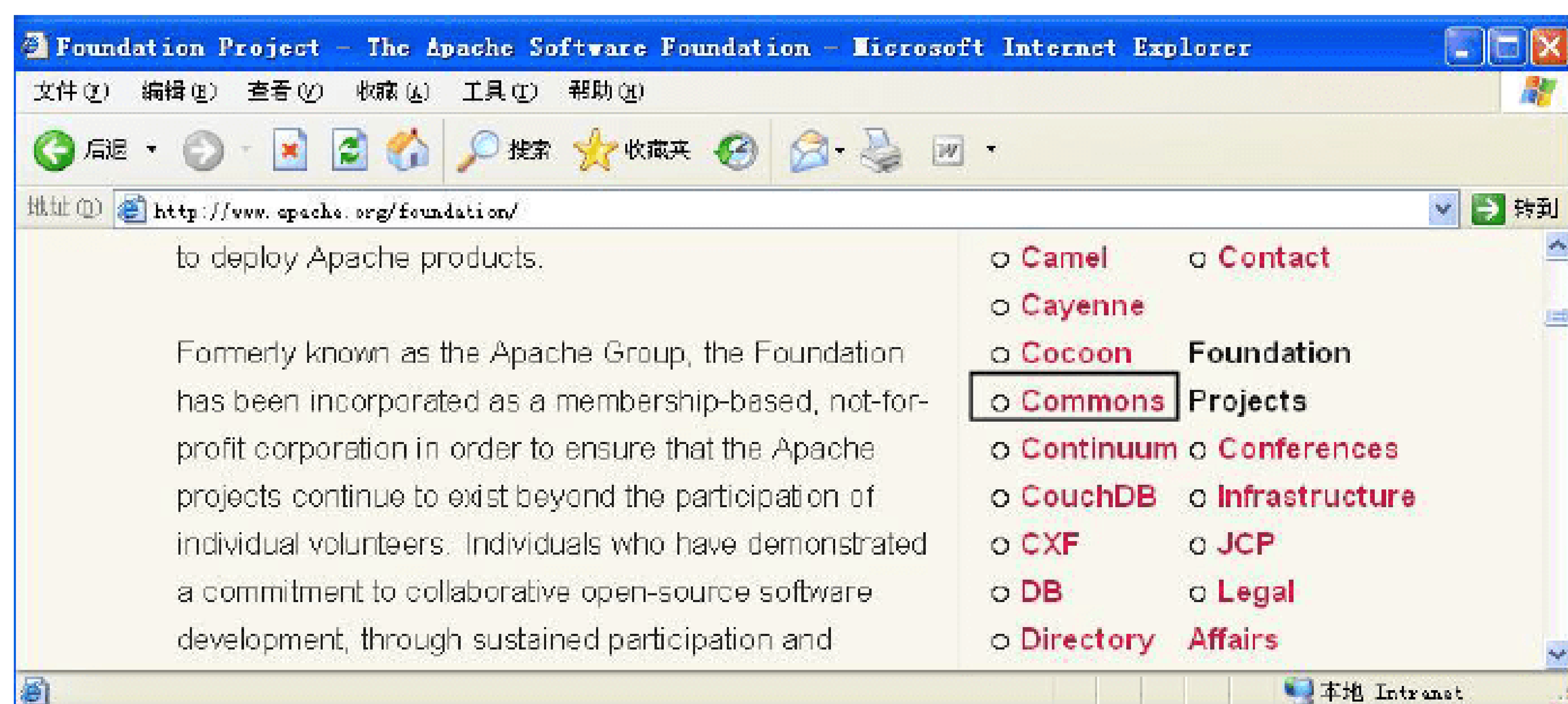


图 18.10　Apache 组织的产品列表

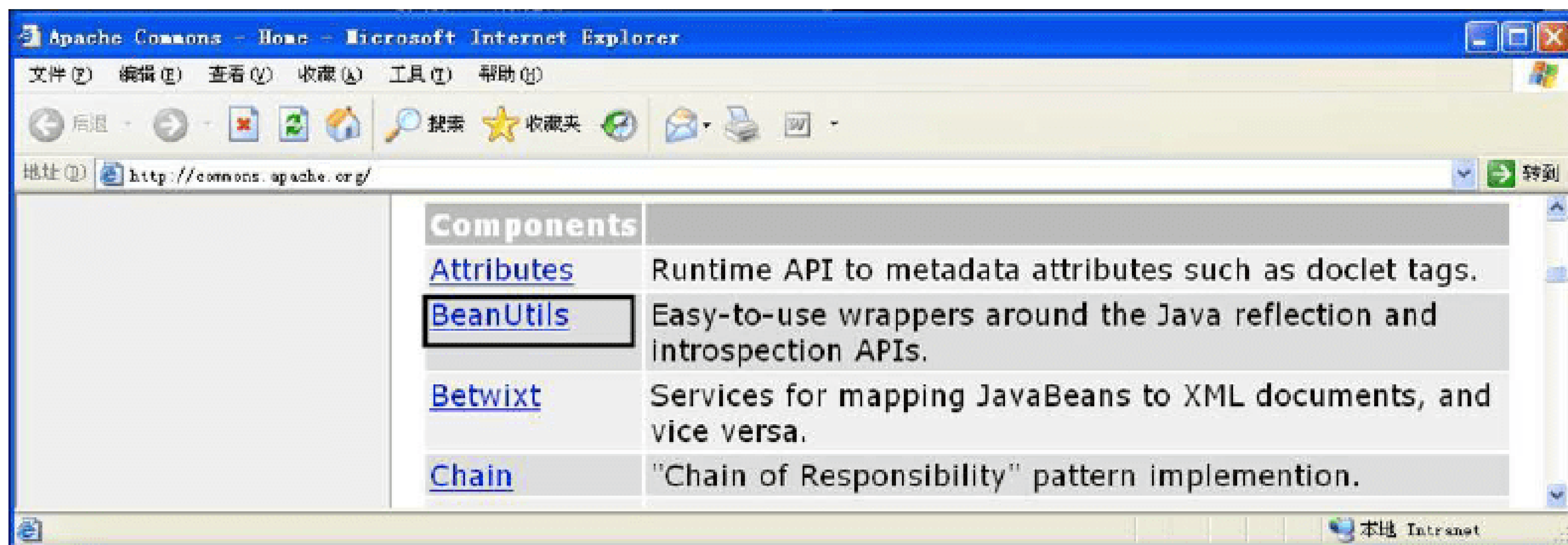


图 18.11　Commons 系列产品列表的页面

（3）在关于 Commons-Beanutils 组件的页面中，单击 Releases 目录中的 here 链接就可以进入关于下载该组件的页面，如图 18.12 所示。

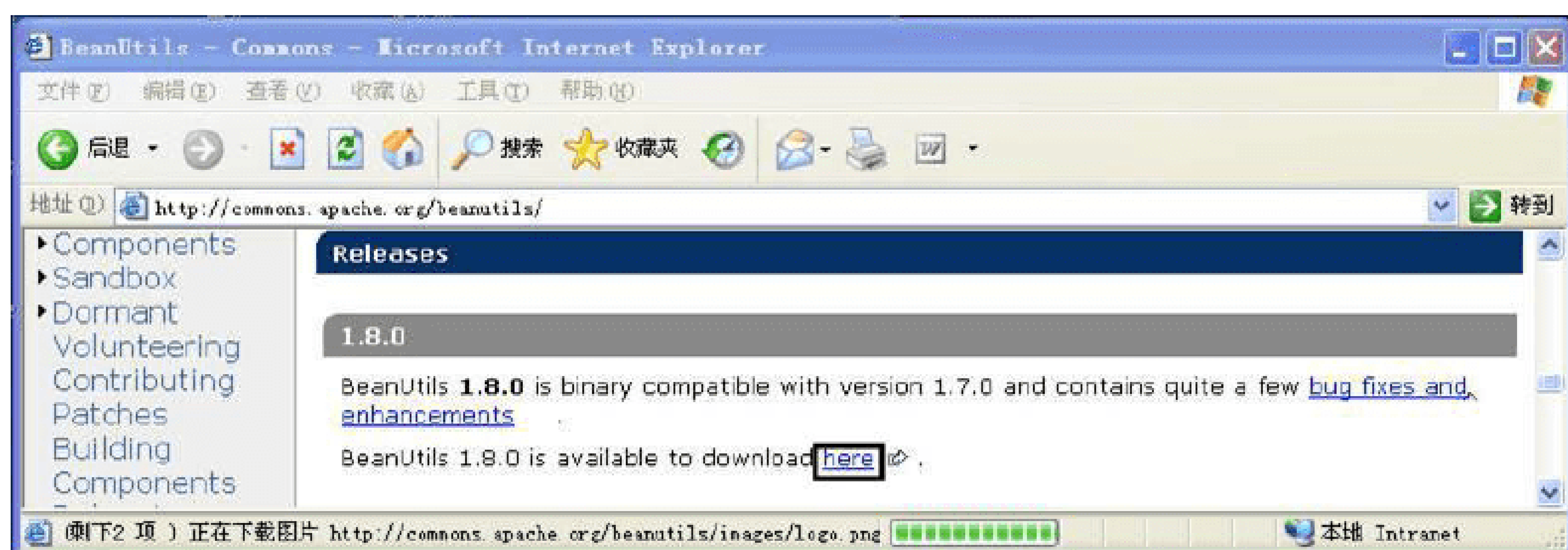


图 18.12　关于 Commons-Beanutils 组件页面

（4）在关于下载 Commons-Beanutils 组件的页面中，单击 Binary 目录中 1.8.0.zip 的链

接就可以实现该组件的下载，如图 18.13 所示。

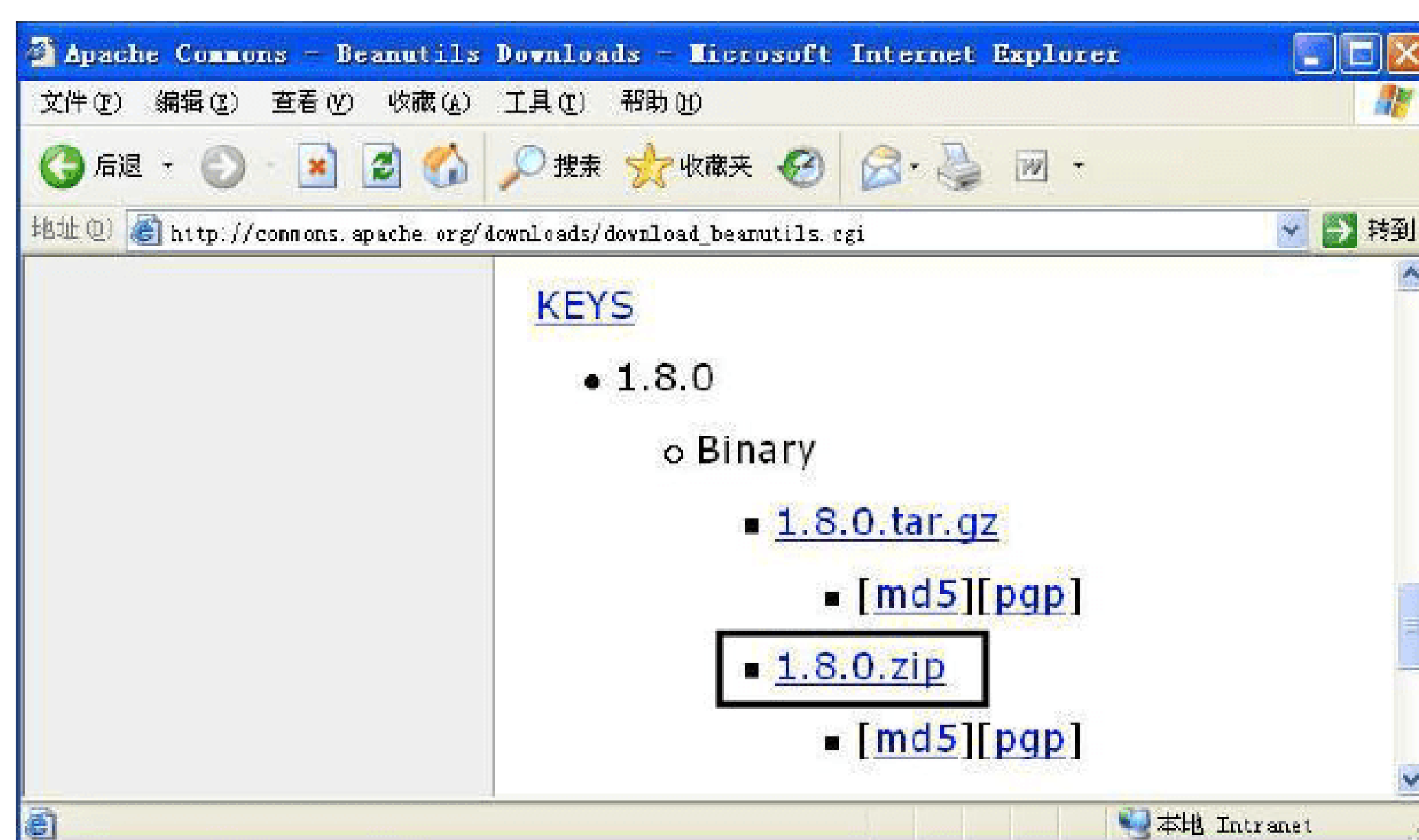


图 18.13　下载 Commons-Beanutils 组件

至此，就完成了对 Commons-Beanutils 组件的下载。

18.2.2　了解 Commons-Beanutils 组件

18.2.1 节介绍了如何下载 Commons-Beanutils 组件，下载完这些组件后就可以在 Java Web 项目中使用该框架。在具体使用之前，解压 commons-beanutils-1.8.0-bin.zip 文件后的具体目录如图 18.14 所示。各个文件的作用如下。

- apidocs：关于 commons-beanutils 组件的帮助文档。
- commons-beanutils-1.8.0.jar：关于 commons-beanutils 组件的 jar 文件。
- commons-beanutils-1.8.0-sources.jar：关于 commons-beanutils 组件的源代码。
- commons-beanutils-core-1.8.0.jar：关于 commons-beanutils 组件的核心 jar 文件。

为了便于使用，可以复制 commons-beanutils-1.8.0.jar 这个 jar 文件到相应的文件夹中，然后为这个 jar 文件在 MyEclipse 开发环境中专门建立一个名为 javabean 的用户库，如图 18.15 所示。

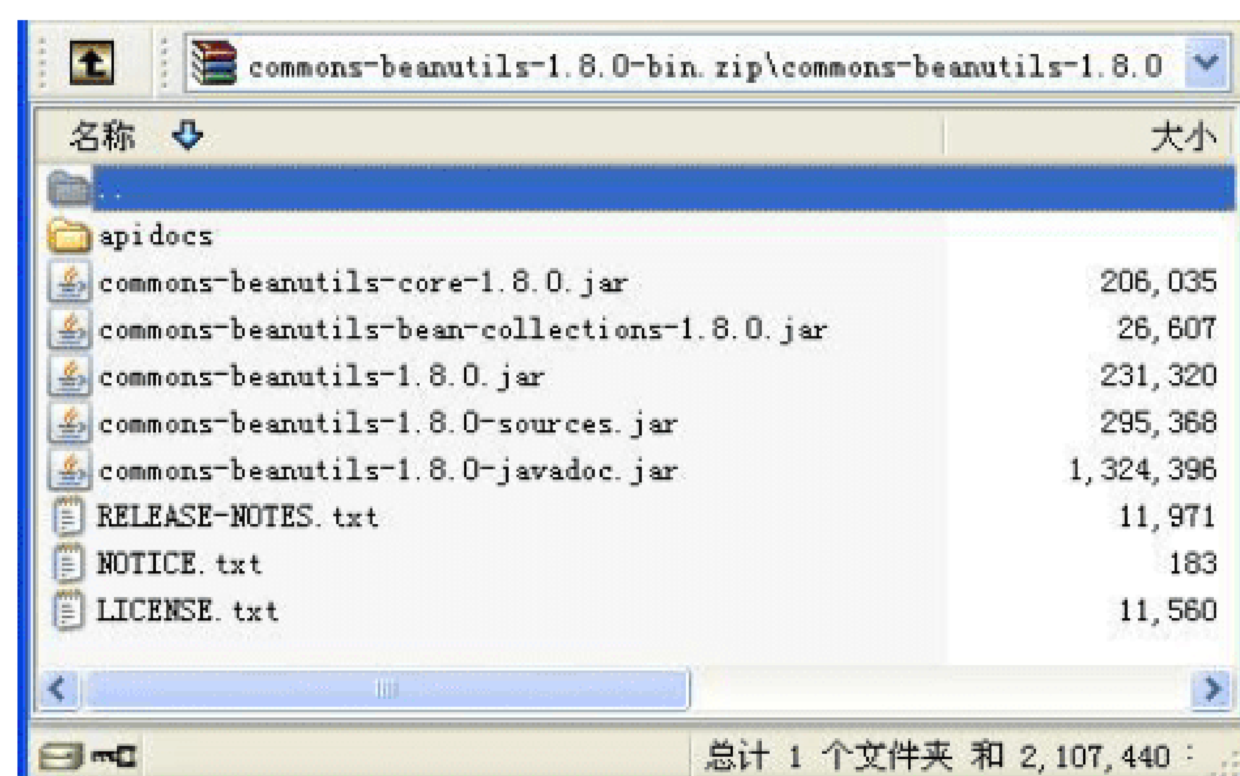


图 18.14　目录结构

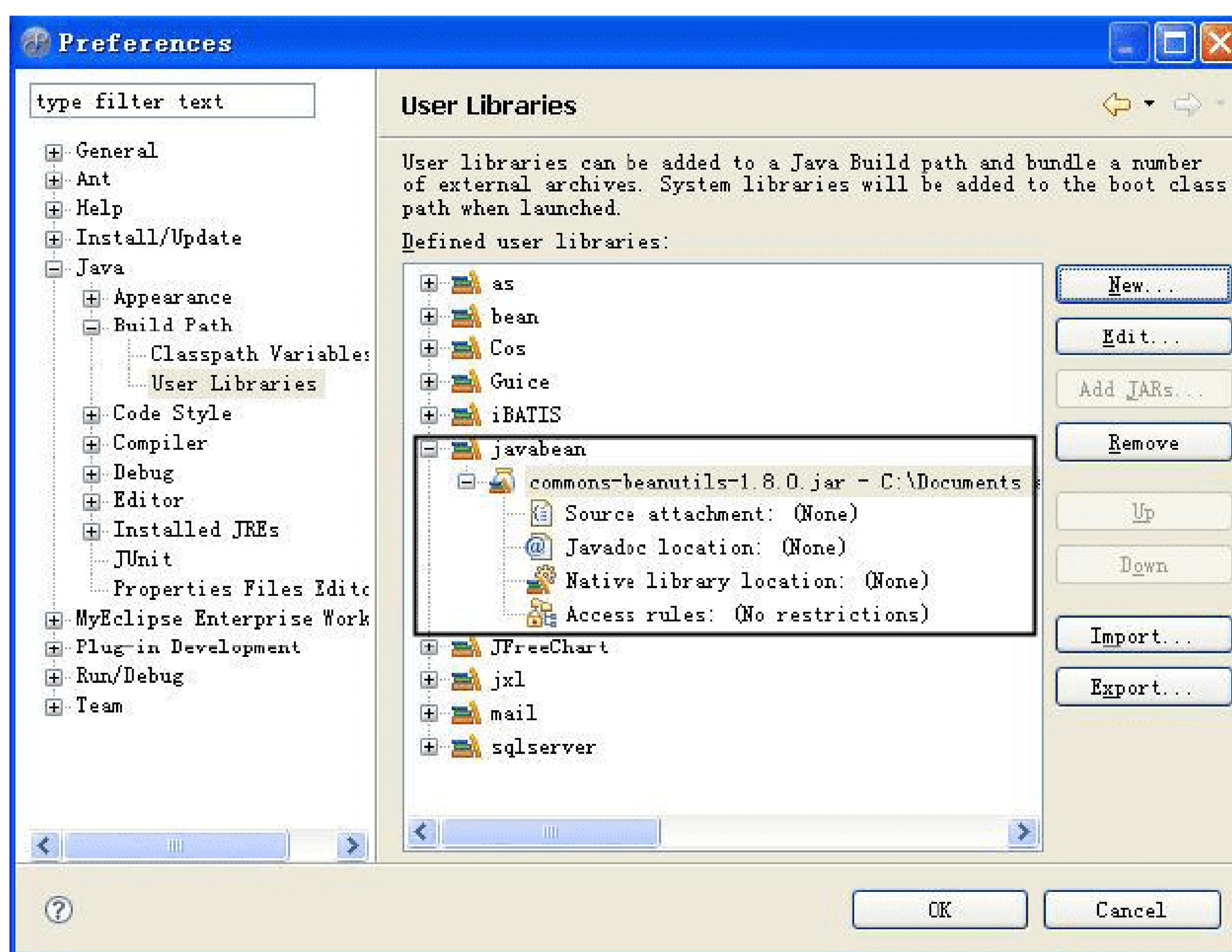


图 18.15　javabean 用户库

至此，就完成了 Commons-BeanUtils 组件在 MyEclipse 开发环境中的配置。

18.2.3　初步学习 Commons-BeanUtils 组件——创建 JavaBean

为了方便讲解 Commons-BeanUtils 组件的用法，本节将创建 3 个 JavaBean：User.java、Profile.java 和 Address.java 作为操作的对象。它们的具体内容分别如代码 18.1、代码 18.2 和代码 18.3 所示。

代码 18.1　User 对象：User.java

```
...
public class User {
    private Long userId;                          //创建 userId 属性
    private String username;                      //创建 username 属性
    private String password;                      //创建 password 属性
    private Profile profile;                      //创建 profile 属性
    //配置相应属性的 get()和 set()方法
    ...
}
```

代码 18.2　Profile 对象：Profile.java

```
...
public class Profile {
    private Map<String, String> phone;            //创建 phone 属性
    private Address[] address;                    //创建 address 属性
    private Date birthDate;                       //创建 birthDate 属性
    private String Email;                         //创建 E-mail 属性
    //配置相应属性的 get()和 set()方法
...
}
```

代码 18.3　Address 对象：Address.java

```
...
public class Address {
    private String postCode;                      //创建属性 postCode
    private String country;                       //创建属性 country
    private String city;                          //创建属性 city
    private String addr;                          //创建属性 addr

    public Address() {                            //无参构造函数
    }
    //有参构造函数
   public Address(String country, String city, String postCode, String addr)
    {
        this.country = country;
        this.city = city;
        this.postCode = postCode;
        this.addr = addr;
    }
    //配置相应属性的 get()和 set()方法
...
}
```

【代码解析】

从上述 3 个 JavaBean 的具体内容中，可以发现 User 对象与 Profile 对象间的关系为一对一，这是因为 User.java 的内容中存在 private Profile profile 代码；而 Profile 对象与 Address 对象之间的关系为一对多，这是因为 Profile.java 的内容中存在 private Address[] address 代码，类型为数组。

18.2.4　初步学习 Commons-BeanUtils 组件——使用相关 API

本节将接着 18.2.3 节的内容创建两个 Java 类：BeanUtilsExample1.java 和 BeanUtils-Example2.java。在这两个类中分别利用 Commons-BeanUtils 组件的相关 API 来操作 18.2.3 节创建的 JavaBean。

BeanUtilsExample1.java 文件主要介绍如何操作和复制属性，该文件的具体内容如代码 18.4 所示。

代码 18.4　操作属性：BeanUtilsExample1.java

```
...
public class BeanUtilsExample1 {
    //初始化数据
    private User prepareData() {
        //创建 profile 对象并对其初始化
        Profile profile = new Profile();     //创建对象 profile
        profile.setEmail("cjgong@126.com"); //设置对象 profile 属性 E-mail
        //设置对象 profile 属性 birthdata
        profile.setBirthDate(new GregorianCalendar(1999, 9, 10).
        getTime());
        //设置对象 profile 属性 phone
        Map<String, String> phone = new HashMap<String, String>();
        phone.put("home", "123456");              //住家电话
        phone.put("xiaolt", "987654");            //小灵通电话
        profile.setPhone(phone);
        //设置对象 profile 属性 address
        Address[] address = { new Address("中国", "南京", "100120", "南京市
        北大街 888 号"),
                new Address("中国", "西安", "100120", "长安区 666 号") };
        profile.setAddress(address);
        //创建 user 对象并对其初始化
        User user = new User();                   //创建对象 user
        user.setUserId(new Long(10000000));       //设置对象 user 属性 userid
        user.setUsername("武小");                 //设置对象 user 属性 username
        user.setPassword("123456789");        //设置对象 user 属性 userpassword
        user.setProfile(profile);                 //设置对象 user 属性 profile
        return user;
    }
    public static void main(String[] args) {
        //创建对象 example
        BeanUtilsExample1 example = new BeanUtilsExample1();
        //调用 prepareData()返回对象 user
        User user = example.prepareData();
        try {
            //通过 BeanUtils 中的 getProperty()方法输出各个属性的值
```

```
System.out.println("输出对象的属性值-------------------");
System.out.println(BeanUtils.getProperty(user, "userId"));
System.out.println(BeanUtils.getProperty(user, "username"));
System.out.println(PropertyUtils.getProperty(user,
"username"));
System.out.println(BeanUtils.getProperty(user, "profile.
email"));
System.out.println(BeanUtils.getProperty(user, "profile.
birthDate"));
System.out.println(BeanUtils.getProperty(user, "profile.
phone(home)"));
System.out.println(BeanUtils.getProperty(user, "profile.
phone(office)"));
System.out.println(BeanUtils.getProperty(user, "profile.
address[0].city"));
System.out.println(BeanUtils.getProperty(user, "profile.
address[0].addr"));

User user2 = new User();                //创建对象 user2
BeanUtils.copyProperties(user2, user); //复制 user 对象到 user2
System.out.println("输出复制属性的属性值-------------------");
//通过 BeanUtils 中的 getProperty()方法输出各个属性的值
System.out.println(BeanUtils.getProperty(user, "username"));
System.out.println(BeanUtils.getProperty(user, "profile.
birthDate"));
System.out.println(BeanUtils.getProperty(user, "profile.
phone(home)"));
System.out.println(BeanUtils.getProperty(user, "profile.
address[0].addr"));

System.out.println("输出复制属性修改以后的属性值-------------");
//通过 BeanUtils 中的 setProperty()方法设置各个属性的值
BeanUtils.setProperty(user2, "userId", new Long(8888888));
PropertyUtils.setProperty(user2, "username", "张小");
BeanUtils.setProperty(user2, "profile.email", "cjgong1@126.
com");
BeanUtils.setProperty(user2, "profile.birthDate", new
GregorianCalendar(1900, 2, 5).getTime());
BeanUtils.setProperty(user2, "profile.address[0]", new
Address("中国", "北京", "600600", "北京大道 111 号"));
//通过 BeanUtils 中的 getProperty()方法获取各个属性的值
System.out.println(BeanUtils.getProperty(user2, "userId"));
System.out.println(BeanUtils.getProperty(user2,
"username"));
System.out.println(BeanUtils.getProperty(user2, "profile"));
System.out.println(BeanUtils.getProperty(user2, "profile.
email"));
System.out.println(BeanUtils.getProperty(user2, "profile.
birthDate"));
System.out.println(BeanUtils.getProperty(user2, "profile.
address[0].city"));

System.out.println("与被复制属性值的对象的比较-------------");
System.out.println(BeanUtils.getProperty(user, "userId"));
System.out.println(BeanUtils.getProperty(user, "username"));
System.out.println(BeanUtils.getProperty(user, "profile"));
System.out.println(BeanUtils.getProperty(user, "profile.
email"));
System.out.println(BeanUtils.getProperty(user, "profile.
```

```
                birthDate"));
                System.out.println(BeanUtils.getProperty(user, "profile.
                address[0].city"));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

运行该程序，则会出现如图 18.16 所示的结果。

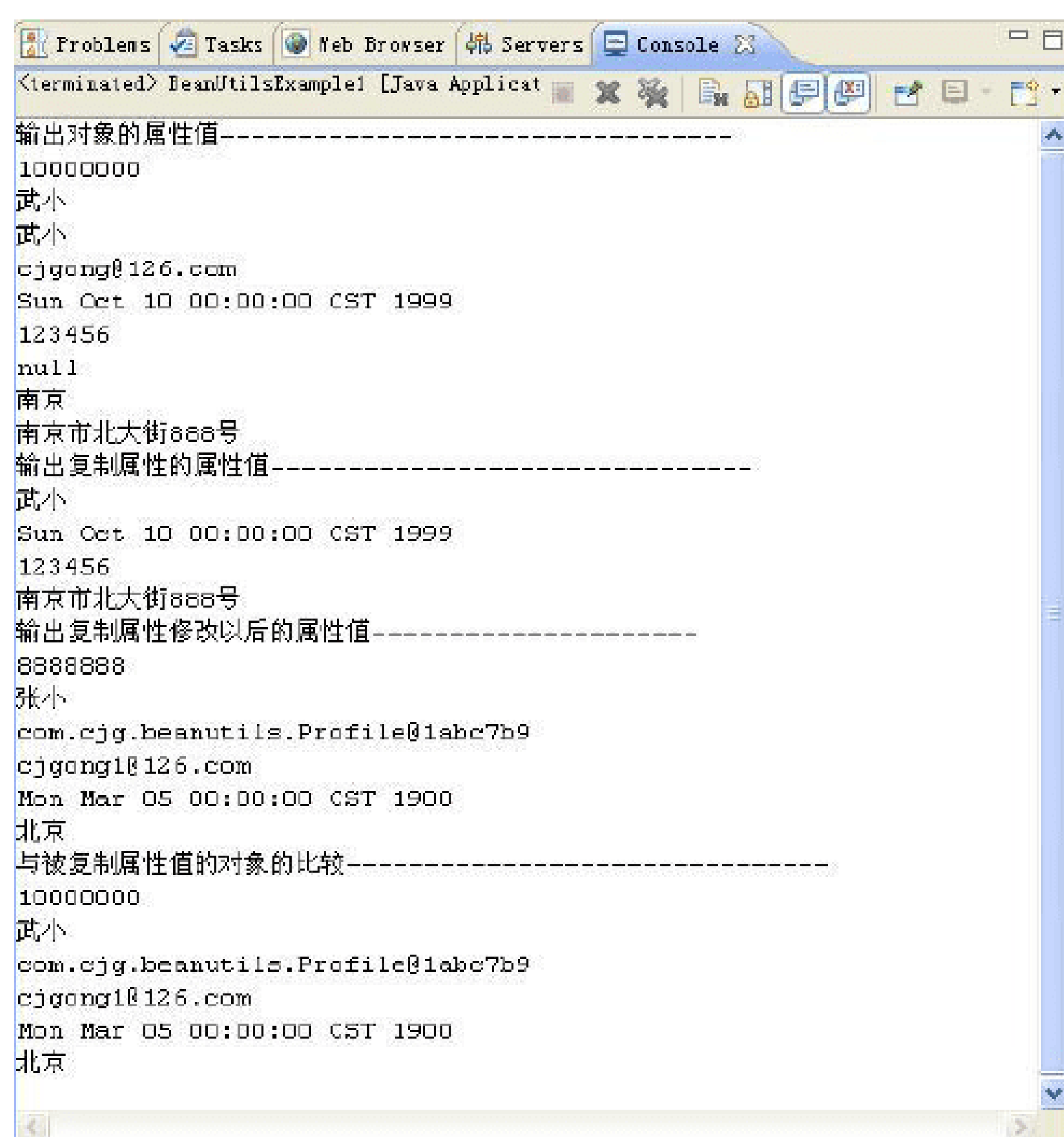


图 18.16　运行结果

【代码解析】

- ❑ 在 BeanUtils 组件中可以通过 getProperty()方法来获取 JavaBean 的属性值，根据开发文档可以发现 Property 被分成 3 种类型：Simple（简单类型）如 Stirng、Int 等；Indexed（索引类型）如数组、arrayList 等；Maped 类型如 Map。
- ❑ 在具体调用 getProperty()方法时，当属性为简单类型时，具体方式如下所示。

```
BeanUtils.getProperty(JavaBean 对象,属性名)
```

例如：BeanUtils.getProperty(user,"userId")。当属性为索引类型时，具体方式如下所示。

```
BeanUtils.getProperty(JavaBean 对象, 属性名[索引值])
```

例如：BeanUtils.getProperty(user,"profile.address[0].city")。当属性为 Maped 类型时，具体方式如下所示。

```
BeanUtils.getProperty(JavaBean 对象, 属性名（key 值）)
```

例如：BeanUtils.getProperty(user,"profile.phone(office)")。

- ❑ 同理 setProperty()方法也具有 3 种类型，具体调用方式与 getProperty()方法也一样，分别为：

```
BeanUtils.setProperty(JavaBean 对象,属性名,属性值)
BeanUtils.setProperty(JavaBean 对象, 属性名[索引值], 属性值)
BeanUtils.setProperty(JavaBean 对象, 属性名（key 值）, 属性值)
```

- ❑ 在 BeanUtils 组件中可以通过 copyProperties()方法来实现 JavaBean 之间的复制。具体方式如下所示。

```
BeanUtils. copyProperties(JavaBean 对象,被复制的 JavaBean 对象)
```

注意： copyProperties()方法主要实现浅复制，即复制后两个 Bean 对象的同一个属性可能拥有同一个对象的 ref。

BeanUtilsExample2.java 文件主要介绍关于 LazyDynaBean 的各种操作，该文件的具体内容如代码 18.5 所示。

代码 18.5　关于 LazyDynaBean 使用：BeanUtilsExample2.java

```
...
public class BeanUtilsExample2 {
    public static void main(String args[]) throws Exception {
        //创建动态 address 的 JavaBean
        LazyDynaBean address = new LazyDynaBean();
        address.set("country", "中国");
        address.set("city", "南京");
        address.set("postCode", "100120");
        address.set("addr", "南京北大街 888 号");
        //创建动态 profile 的 JavaBean
        LazyDynaBean profile = new LazyDynaBean();
        profile.set("phone", "home", "123456");
        profile.set("phone", "xiaolt", "987654");
        profile.set("email", "cjgong@126.com");
        profile.set("address", 0, address);
        profile.set("birthDate", new GregorianCalendar(1900, 2, 5).
        getTime());
        //创建动态 user 的 JavaBean
        LazyDynaBean user = new LazyDynaBean();
        user.set("userId", new Long(123456789));
        user.set("username", "常小");
        user.set("password", "123456");
        user.set("profile", profile);
        //输出属性
        System.out.println(BeanUtils.getProperty(user, "username"));
        System.out.println(BeanUtils.getProperty(user, "profile.
        birthDate"));
        System.out.println(BeanUtils.getProperty(user,  "profile.address
        [0].addr"));
        System.out.println(BeanUtils.getProperty(user, "profile.phone
        (office)"));
    }
}
```

运行该程序，则会出现如图 18.17 所示的结果。

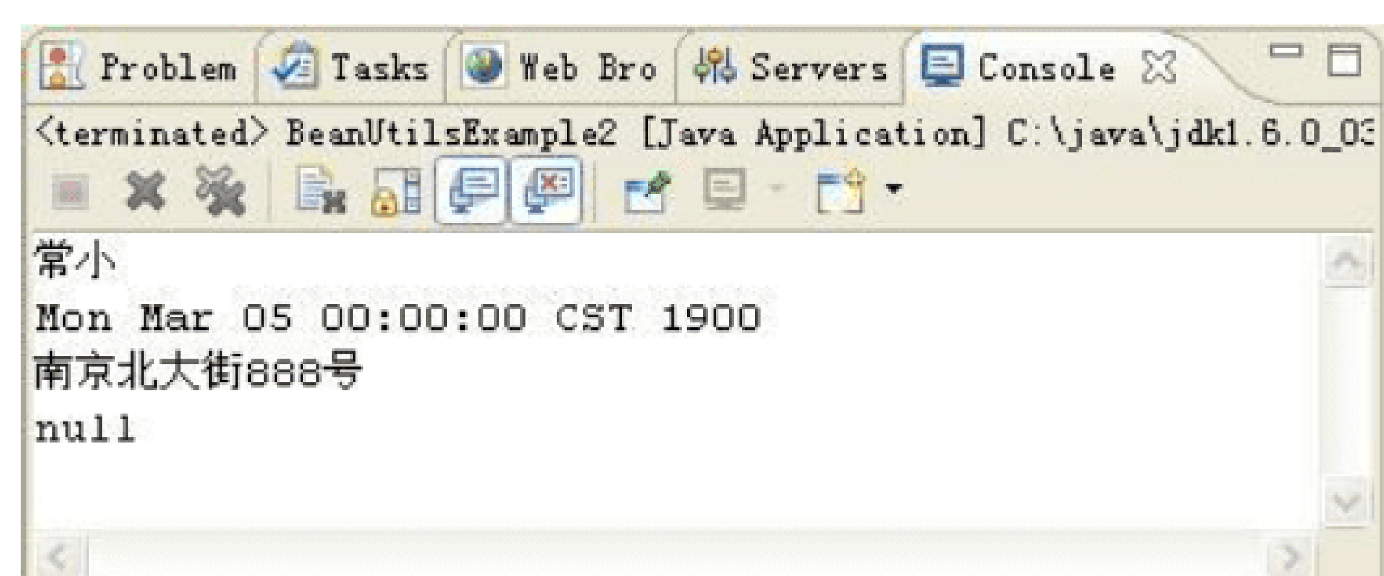


图 18.17　运行结果

【代码解析】

- 在 BeanUtils 组件中可以通过 LazyDynaBean 来创建一个动态的 JavaBean，之所以要使用动态 JavaBean，因为可以通过不创建标准的 JavaBean，而直接对动态 JavaBean 进行操作。
- 在 LazyDynaBean 中，不仅支持 3 种类型的属性，而且还存在两个非常重要的字段：returnnull 和 restricted。字段 restricted 默认值为 false，即当对象中调用 getProperty() 方法时，若该 Bean 对象中没有此字段，则返回 null。当字段 restricted 值为 true 时，则会自动增加一个字段。例如如下代码：

```
BeanUtils.getProperty(user, "profile.phone(office)")
```

- 为了防止改变 LazyDynaBean 的属性，可以设置字段 restricted 的值为 true。由于 LazyDynaBean 的属性值默认为 false，所以可以增删修改字段。

18.3　关于 Hibernate 框架中一些通用类

在 Hibernate 框架中经常会重复一些操作，如果每次需要时都要编写代码，那么将使代码显的不友好。本节为了巩固 Hibernate 框架的一些常用知识，将对 Hibernate 框架中的一些常见操作进行封装而创建出一些工具类。

18.3.1　操作常见对象的通用类

Hibernate_Util 工具类不仅实现操作 SessionFactory 对象，而且还实现关于 Session 对象和 Transaction 对象的各种操作。该类的具体内容如代码 18.6 所示。

代码 18.6　操作常见对象：Hibernate_Util.java

```
...
public class Hibernate_Util {
    //创建静态变量
    private static final String config_file = "hibernate.cfg.xml";
                                          //创建关于配置文件的常量
    private static final Configuration conf = new Configuration();
                                          //创建 Configuration 对象
    private static SessionFactory factory = null;//创建 SessionFactory 对象
    //生成日记对象
    private static final Log log = LogFactory.getLog(Hibernate_Util.class);
    // 盛放 Session 对象的 sessThreadLocal 变量
    private   static   final   ThreadLocal<Session>   sessThreadLocal   =   new
    ThreadLocal<Session>();
    //盛放 transacion 对象的 tranThreadLocal 变量
    private  static  final  ThreadLocal<Transaction>  tranThreadLocal  =  new
    ThreadLocal<Transaction>();
    public Hibernate_Util() {                     //创建构造函数
    }
    static {                                      //静态模块
        conf.configure(config_file);
        factory = conf.buildSessionFactory();
    }
    //实现对 Session 操作
    public static Session getSession() {              //创建 Session 对象
        Session session = sessThreadLocal.get();      //当 Session 对象存在
        //当 Session 对象不存在
        try {
            if (session == null) {
                if (factory == null)
                    //获取 factory 对象
                    factory = conf.buildSessionFactory();
                session = factory.openSession();      //获取 Session 对象
                log.info("Session 创建成功！");        //生成日记
                sessThreadLocal.set(session);         //保存 Session 对象
            }
        } catch (Exception e) {
            e.printStackTrace();
```

```
            log.error("Session 创建失败！");
        }
        return session;
    }
    public static void closeSession() {                  //实现关闭 Session 对象
        Session session = sessThreadLocal.get();         //获取 Session 对象
        try {
            if (session != null) {                       //当 Session 对象不为空
                session.close();                         //关闭 Session 对象
                log.info("Session 关闭成功！");
                sessThreadLocal.set(null);               //取消 Session 存储
            }
        } catch (Exception e) {
            e.printStackTrace();
            log.error("Session 关闭失败！");
        }
    }
    //实现对事务操作
    public static void beginTran() {                     //实现开始事务功能操作
        Session session = getSession();                  //获取 Session 对象
        try {
            if (session != null) {                       //当 Session 对象不存在时
                //开始事务
                Transaction tran = session.beginTransaction();
                log.info("事务开始成功！");              //生成日记
                tranThreadLocal.set(tran);               //存储事务
            }
        } catch (Exception e) {
            e.printStackTrace();
            log.error("事务开始失败！");
        }
    }
    public static void commitTran() {                    //实现提交事务功能操作
        Transaction tran = tranThreadLocal.get();        //获取事务对象
        try {
            if (tran != null && !tran.wasCommitted() && !tran.
            wasRolledBack()) {
                tran.commit();                           //提交事务
                log.info("事务提交成功！");              //生成日记
                tranThreadLocal.set(null);               //取消事务的存储
            }
        } catch (Exception e) {
            e.printStackTrace();
            log.error("事务提交失败！");
        }
    }
    public static void rollbackTran() {                  //实现回滚事务功能操作
        Transaction tran = tranThreadLocal.get();        //获取事务对象
        try {
            if (tran != null && !tran.wasCommitted() && !tran.was-
            RolledBack()) {
                tran.rollback();                         //实现事务回滚
                log.info("事务回滚成功！");              //生成日记
                tranThreadLocal.set(null);               //取消事务的存储
            }
        } catch (Exception e) {
            e.printStackTrace();
```

```
                log.error("事务回滚失败！");
            }
        }
}
```

【代码解析】

- 在上述代码中，首先会初始化 6 个静态常量。6 个变量之所以会是静态常量，除了需要在静态模块中为 conf 常量和 factory 常量赋值，最主要的还是在生成该类对象时，这 6 个常量可以常驻内存。
- 在操作 Session 对象的方法中，通过方法 getSession()获取 Session 对象，通过方法 closeSession()关闭 Session 对象。
- 在操作 Transaction 对象的方法中，通过方法 beginTran()开启事务，通过方法 commitTran()提交事务，通过方法 rollbackTran()回滚事务。

18.3.2　实现各种方法的模板类

Hibernate_Template 类通过调用工具类 Hibernate_Util 中的相关方法，来实现关于 Hibernate 框架的各种操作。该类的具体内容如代码 18.7 所示。

代码 18.7　模板类：Hibernate_Template.java

```
...
public class Hibernate Template {
    public Hibernate_Template() {                         //创建构造函数
    }
    public Session getSession() {                         //关于 Session 对象操作
        return Hibernate Util.getSession();
    }
    public void closeSession() {
        Hibernate Util.closeSession();
    }
    public void beginTran() {                             //关于事务对象操作
        Hibernate Util.beginTran();
    }
    public void commitTran() {
        Hibernate_Util.commitTran();
    }
    public void rollbackTran() {
        Hibernate Util.rollbackTran();
    }
    public Object get(Class pocalss, Serializable id) {    //得到一个对象
        return this.getSession().get(pocalss, id);
    }
    public Object load(Class pocalss, Serializable id) {
        return this.getSession().load(pocalss, id);
    }
    public Object get(String hql, Object... params) {
        Query query = this.getSession().createQuery(hql);
        this.setParameter(query, params);
        return query.uniqueResult();
    }
    public Integer count(String hql, Object... params) {//获取总的对象个数
        Object obj = this.get(hql, params);
```

```
        return (Integer) obj;
    }
    public List getObjects(String hql, Object... params) {
                                                    //把获取的对象封装到 List
        Query query = this.getSession().createQuery(hql);
        this.setParameter(query, params);
        return query.list();
    }
    public List getObjects(String hql, int curpage, int pagerecord,
            Object... params) {
        Query query = this.getSession().createQuery(hql);
        this.setParameter(query, params);
        query.setFirstResult((curpage - 1) * pagerecord);      //起点位置
        query.setMaxResults(pagerecord);              //抓取的数量
        return query.list();
    }
    public List getObjects(String hql, int curpage, int pagerecord,
            Class voclass, Object... params) {
        List vos = new ArrayList();//vo 对象
        try {
            Query query = this.getSession().createQuery(hql);
            this.setParameter(query, params);
            query.setFirstResult((curpage - 1) * pagerecord);  //起点位置
            query.setMaxResults(pagerecord);           //抓取的数量
            List<Object[]> list = query.list();
            for (Object[] objs : list)//emp,dept
            {
                Object vo = voclass.newInstance();
                for (Object obj : objs) {
                    BeanUtils.copyProperties(vo, obj);
                }
                vos.add(vo);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return vos;
    }
    public List getObjects(DetachedCriteria c, int curpage, int pagerecord,
            Criterion... params) {
        if (params != null && params.length > 0) {
            for (Criterion cond : params) {
                c.add(cond);
            }
        }
        //传递一个 Session 由 DetachedCriteria 得到可以执行查询的 Criteria
        Criteria criteria = c.getExecutableCriteria(this.getSession());
    criteria.setFirstResult((curpage - 1) * pagerecord);
        criteria.setMaxResults(pagerecord);
        criteria.setResultTransformer(criteria.DISTINCT_ROOT_ENTITY);
        return criteria.list();
    }
  public Integer countByDetached(DetachedCriteria c, Criterion... params)
    {
        if (params != null && params.length > 0) {
            for (Criterion cond : params) {
                c.add(cond);
            }
        }
        //传递一个 Session 由 DetachedCriteria 得到可以执行查询的 Criteria
```

```
        Criteria criteria = c.getExecutableCriteria(this.getSession());
    criteria.setResultTransformer(criteria.DISTINCT ROOT ENTITY);
        criteria.setProjection(Projections.rowCount());
        return (Integer) criteria.uniqueResult();
    }
    public void delete(Class pocalss, Serializable id) throws Exception
    {   //无参删除一个对象
        try {
            Object obj = this.getSession().get(pocalss, id);
            this.getSession().delete(obj);
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        }
    }
    public void delete(Object po) throws Exception {        //有参删除方法
        try {
            this.getSession().delete(po);
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        }
    }
    public void delete(String hql) throws Exception {       //有参删除方法
        try {
            Query query = this.getSession().createQuery(hql);
            query.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        }
    }
    public void update(Object po) throws Exception {        //修改一个对象
        try {
            this.getSession().update(po);
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        }
    }
    public void update(String hql, Object... params) throws Exception {
        try {
            Query query = this.getSession().createQuery(hql);
            this.setParameter(query, params);
            query.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        }
    }
    public void merge(Object po) throws Exception {         //合并属性方法
        try {
            this.getSession().merge(po);
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        }
    }
    public Serializable save(Object po) throws Exception {//实现保存操作
        try {
```

```
            return this.getSession().save(po);
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        }
    }
    public void saveOrUpdate(Object po) throws Exception {
                                                    //实现保存和更新操作
        try {
            this.getSession().saveOrUpdate(po);
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        }
    }
    public void persist(Object po) throws Exception {  //实现持久化操作
        try {
            this.getSession().persist(po);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void setParameter(Query query, Object... params) {
                                                        //设置参数值方法
        if (params != null && params.length > 0) {
            for (int i = 0; i < params.length; i++) {
                query.setString(i, params[i].toString());
            }
        }
    }
}
```

为了便于其他各层能够使用 Hibernate_Template 类中的某个方法，可以对该类进行封装。HibernateDaoSupport 类实现对模板类的封装，该类的具体内容如代码 18.8 所示。

代码 18.8　封装模板类：HibernateDaoSupport.java

```
...
public class HibernateDaoSupport
{
    private Hibernate Template hibernate Template;
                                            //创建 Hibernate_Template 属性
    public HibernateDaoSupport()            //创建构造函数
    {
      this.setHibernate Template(new Hibernate Template());
    }
    public Hibernate Template getHibernate Template() {
                                            //配置 Hibernate_Template 属性
        return hibernate Template;
    }
    public void setHibernate_Template(Hibernate_Template hibernate_
    Template) {
        this.hibernate_Template = hibernate_Template;
    }
}
```

至此，完成了对 Hibernate 框架各种操作的封装。

18.3.3　关于页面信息类

如果想实现 Hibernate 分页系统，首先必须要创建一个关于页面信息的 JavaBean 类，在该类中封装了关于分页的所有信息，具体内容如代码 18.9 所示。

代码 18.9　页面信息类：Pageinfo.java

```
...
public class Pageinfo{
    private int curpage=1;                          //创建 curpage 属性
    private int allpage;                            //创建 allpage 属性
    private int allrecord;                          //创建 allrecord 属性
    private int pagerecord;                         //创建 pagerecord 属性
    private int nextpage;                           //创建 nextpage 属性
    private int previouspage;                       //创建 previouspage 属性
    private List pagedata;                          //创建 pagedata 属性
    public Pageinfo()                               //无参构造函数
    {
    }
    public Pageinfo(int allrecord,int pagerecord,List pagedata)
                                                    //有参构造函数
    {
        this.allrecord=allrecord;
        this.pagerecord=pagerecord;
        this.pagedata=pagedata;
        this.allpage=(allrecord+pagerecord-1)/pagerecord;
    }
    //省略属性 allpage、allrecord、pagerecord、nextpage、previouspage 和
    pagedata 的配置
...
}
```

【代码解析】

- 首先创建分页所需要的各种属性：curpage 属性表示当前第几页；allpage 属性表示所有页数；allrecord 属性表示所有记录数；pagerecord 属性表示分页单位即每页记录数；nextpage 属性表示下一页；previouspage 属性表示上一页；pagedata 属性表示一页数据。
- 在创建有参构造函数时，之所以需要传进来 3 个参数而不是 7 个参数，是因为通过该 3 个参数就可以计算出其他 4 个参数。

18.4　实现 Hibernate 分页系统前期准备

本节除了将详细介绍如何设计关于 Hibernate 分页系统的数据库和表外，还将配置实现该系统将利用的 Hibernate+MySQL 框架环境。其中 Hibernate 框架的版本号为 Hibernate 3.0，数据库 MySQL 为 MySQL 5.0。

18.4.1　数据库设计

Hibernate 分页系统需要建立一个数据库和在该数据库中建立一张表，存放表的数据库 page 和存放部门信息的表 dept。

1．创建数据库page

如果想创建出数据库 page，可以在 MySQL 的命令行窗口中输入如下命令：

```
CREATE DATABASE 'page'
```

2．创建表格dept

如果想创建出表 dept，可以在 MySQL 的命令行窗口中输入相应命令。该表的相关字段如表 18.1 所示，该表的模拟数据如图 18.18 所示。

表 18.1　表dept信息

字 段 名 称	数 据 类 型	字 段 说 明	字 段 名 称	数 据 类 型	字 段 说 明
id	int(11)	编号	no	int(11)	部门号码
name	varchar(50)	部门名称	descn	varchar(50)	部门描述

	id	name	no	descn
	2	2	2	2
	3	3	3	3
	4	5	5	5
	5	6	6	6
	6	6	6	6
	7	9	9	9

图 18.18　模拟数据

18.4.2　关于 Hibernate 3.0 框架的准备

在引入 Hibernate 3.0 框架时，对于其中的 Hibernate 3.0 框架除了需要引入相应的 jar 文件外，还必须得对 hibernate.cfg.xml 文件做相应的配置。

1．引入jar文件

首先引入关于 Hibernate 3.0 框架的核心包：Hibernate3.0.jar、log4j-1.2.13.jar、cglib-nodep-2.1_3.jar、dom4j-1.6.1.jar、commons-collections.jar、c3p0-0.9.0.4.jar、jta.jar 和 antlr-2.7.6.jar。

2．创建hibernate.cfg.xml文件

首先通过 MyEclipse 开发环境中关于 Hibernate 框架的向导让 Hibernate 分页系统支持 Hibernate 3.0 框架，然后通过该开发环境的反向工程向导实现对数据库 Pgae 中的 dept 表进行关系数据库到对象的映射。关于这些持久化类和映射文件可以在具体实现部分查看，它们分别为 Dept.java 和 Dept.hbm.xml。hibernate.cfg.xml 文件的具体内容如代码 18.10 所示。

代码 18.10　实现 hibernate.cfg.xml 文件：hibernate.cfg.xml

```
...
<hibernate-configuration>
    <session-factory>
        <!-- 指定连接数据库方言-->
```

```
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <!-- 指定连接数据库 URL-->
        <property name="connection.url">jdbc:mysql://localhost:3306/page
        </property>
        <!-- 指定连接数据库用户->
        <property name="connection.username">root</property>
        <!-- 指定连接数据库密码-->
        <property name="connection.password">root</property>
        <!-- 指定连接数据库驱动类-->
        <property name="connection.driver class">
            com.mysql.jdbc.Driver
        </property>
        <property name="myeclipse.connection.profile">MySQL</property>
        <!-- 指定 Hibernate 映射文件 -->
        <mapping resource="com/cjg/po/Dept.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

至此，就完成对 Hibernate 分页系统中 Hibernate 3.0 框架的配置。

18.4.3　由反向工程生成的领域模型对象

对数据库 page 中的表格 dept 进行反向工程后，就会自动生成与该表格相对应的领域模型对象和映射文件。实现部门模型的内容如代码 18.11 所示，该模型文件的映射文件内容如代码 18.12 所示。

代码 18.11　部门对象：Dept.java

```
...
public class Dept  implements java.io.Serializable {
    //创建各种属性
    private Integer deptid;                         //创建属性 deptid
    private String deptname;                        //创建属性 deptname
    private Integer deptnum;                        //创建属性 deptnum
    private String deptdesc;                        //创建属性 deptdesc
    public Dept() {                                 //构造函数
    }
    //省略属性 deptid、deptname、deptnum 和 deptdesc 的配置
    ...
}
```

代码 18.12　部门对象映射文件：Dept.hbm.xml

```
...
<hibernate-mapping>
   <!--制定要持久化的类与对应数据库的表映射关系-->
   <class name="org.cjg.po.Dept" table="dept" catalog="page">
      <id name="deptid" type="java.lang.Integer">
         <column name="id" />
         <generator class="increment" />
      </id>
      <!--表 dept 字段 name 与 deptname 属性的映射关系>
      <property name="deptname" type="java.lang.String">
```

```
            <column name="name" length="20" not-null="true" />
        </property>
        <!--表 dept 字段 no 与 deptnum 属性的映射关系>
        <property name="deptnum" type="java.lang.Integer">
            <column name="no" />
        </property>
        <!--表 dept 字段 descn 与 deptdesc 属性的映射关系>
        <property name="deptdesc" type="java.lang.String">
            <column name="descn" length="50" />
        </property>
    </class>
</hibernate-mapping>
```

注意：Dept.java 类可以参考数据库中的表格 dept 来编写，而该类的映射文件则可以通过向导实现。

至此，就完成对 Hibernate 分页系统中 POJO 层的实现。

18.5　关于 Hibernate 分页系统的具体实现

为了让读者可以快速理解和掌握 Hibernate 分页系统功能，在具体讲解时对该功能按照 MVC 模式分成领域模型层、持久层、业务层和表现层。由于在前面章节已经介绍了领域模型层，所以本节将介绍关于 dept 表的持久层、业务层和服务层。

18.5.1　dept 模型对应的持久层

对于 dept 模型对象，在持久层中主要用来实现操作该对象的功能。由于该层采用 DAO 模式来设计，所以创建了两个类：DeptDao.java 和 DeptDaoi.java。

1．编写操作dept模型接口类

DeptDao.java 文件主要用来定义操作 dept 模型对象的方法，具体内容如代码 18.13 所示。

代码 18.13　操作 dept 对象接口类：DeptDao.java

```
...
public interface DeptDao
{
  Serializable save(Dept dept) throws Exception;              //添加记录方法
  List getDepts(int curpage,int pagerecord,String cond);    //获取一页数据
  int count(String cond);                                     //总记录总数
}
```

【代码解析】

- ❑ 在 getDepts()方法中，参数 curpag 表示当前页数；参数 pagerecord 表示分页单位，即每页记录数，参数 cond 表示条件。该方法用来获取一页数据。
- ❑ 在 count()方法中，参数 cond 表示条件，该方法主要用来实现获取总记录数。

2．编辑继承DeptDao接口的类

DeptDaoi.java 类继承了 DeptDao.java 类，实现了操作 dept 模型对象的各个方法，具体内容如代码 18.14 所示。

代码 18.14　操作 dept 对象实现类：DeptDaoi.java

```
...
public class DeptDaoi extends HibernateDaoSupport implements DeptDao {
    public DeptDaoi()                                         //构造函数
    {
    }
    public Serializable save(Dept dept) throws Exception    //添加记录的方法
    {
        Serializable id=this.getHibernate Template().save(dept);
        return id;
    }
    public List getDepts(int curpage, int pagerecord, String cond)
                                                          //获取一页记录方法
    {
        String hql="from Dept dept where 1=1"+cond;
        List  list=this.getHibernate_Template().getObjects(hql,  curpage,
            pagerecord);
        return list;
    }
    public int count(String cond) {                       //获取记录总数目方法
        String hql="select count(dept) from Dept dept where 1=1 "+cond;
        return this.getHibernate Template().count(hql);
    }
}
```

【代码解析】

上述代码中，在具体编写 getDepts() 和 count() 方法时，都通过调用工具类 HibernateDaoSupport 来实现。

18.5.2　dept 模型对象对应的服务层

对于 dept 模型对象，在服务层中主要用来实现操作该对象的功能。由于该层采用 DAO 模式来设计，所以创建了两个类：DeptService.java 和 DeptServiceImpl.java。

1．编写操作dept模型对象接口类

DeptService.java 文件主要用来定义操作 dept 模型对象的方法，具体内容如代码 18.15 所示。

代码 18.15　操作 dept 对象接口类：DeptService.java

```
...
public interface DeptService
{
   int pagerecord=2;                                  //分页单位
   void save(DeptForm form);                          //添加记录
```

```
    Pageinfo getDepts(int curpage,String cond);          //获取 Pageinfo 对象
}
```

【代码解析】

在上述代码中，之所以要在业务层定义分页单位，是因为在一些系统中，每个模块的分页单位有可能不同。

2. 编辑继承DeptDao接口的类

DeptServiceImpl.java 类继承了 DeptService.java 类，实现了操作 dept 模型对象的各个方法，具体内容如代码 18.16 所示。

代码 18.16　操作 dept 对象实现类：DeptServicei.java

```
...
public class DeptServicei extends HibernateDaoSupport implements
DeptService
{
    private DeptDao dao;                                 //创建属性 dao
    public DeptDao getDao() {                            //配置属性 dao
        return dao;
    }
    public void setDao(DeptDao dao) {
        this.dao = dao;
    }
    public DeptServicei()                                //构造函数
    {
        this.setDao(new DeptDaoi());
    }
    public void save(DeptForm form)                      //添加记录方法
    {
        Dept dept=new Dept();                            //创建 Dept 对象
        try{
        this.getHibernate_Template().beginTran();        //开启事务
        BeanUtils.copyProperties(dept,form);             //复制对象
        this.getDao().save(dept);                        //保存记录
        this.getHibernate_Template().commitTran();       //提交事务
        }catch(Exception e){
        e.printStackTrace();
        this.getHibernate_Template().rollbackTran();     //回滚事务
      }
      this.getHibernate_Template().closeSession();       //关闭 Session
    }

    public Pageinfo getDepts(int curpage, String cond) //获取 Pageinfo 对象
    {
        //调用 DAO 层的 getDepts()方法
        List list=this.getDao().getDepts (curpage, pagerecord, cond);
        int count=this.getDao().count(cond);     //调用 DAO 层的 count()方法
        Pageinfo pageinfo=new Pageinfo(count,pagerecord,list);
                                                     //创建出 Pageinfo 对象
        pageinfo.setCurpage(curpage);                //设置当前页
        this.getHibernate_Template().closeSession();    //关闭 Session 对象
        return pageinfo;
    }
}
```

【代码解析】

上述代码中，在具体编写 getDepts()方法时，首先通过调用 DAO 层的 getDepts()方法获取一页数据 list，然后再调用 DAO 层的 count()方法获取所有记录数 count，最后通过调用工具类 Pageinfo 就可以创建出对象 pageinfo。

18.6　关于 Hibernate 分页系统的表示层

为了让读者可以快速理解和掌握 Hibernate 分页系统功能，在具体讲解时对该功能按照 MVC 模式分成为领域模型层、持久层、业务层和表现层。由于在前面章节中已经介绍了除表现层外的其他层，所以本节将详细介绍该系统的表现层。

18.6.1　实现页面的跳转

Hibernate 分页系统中所有请求都由名为 DeptServlet 的 Servlet 类来处理，该类的具体内容如代码 18.17 所示。

代码 18.17　操作 dept 对象实现类：DeptServlet.java

```
...
public class DeptServlet extends HttpServlet {
    private DeptService service;                    //创建属性 Service
    public DeptServlet()                            //创建构造函数
    {
        this.setService(new DeptServicei());
    }
    public DeptService getService() {               //配置属性 Service
        return service;
    }
    public void setService(DeptService service) {
        this.service = service;
    }
    //编写 doGet()方法
    public void doGet(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        this.doPost(request, response);             //调用 doPost()方法
    }
    //编写 doPost()方法
    public void doPost(HttpServletRequest request, HttpServletResponse
    response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=GBK");  //设置编码格式
        String m=request.getParameter("m");         //获取参数 m 的值
        if("save".equals(m)){                       //判断 m 的值
            this.save(request, response);           //调用 save()方法
        }else if("list".equals(m)){
            this.list(request, response);           //调用 list()方法
        }
    }
    //编写保存记录方法
```

```
    public void save(HttpServletRequest request, HttpServletResponse
    response)
        throws ServletException, IOException
    {
        DeptForm form=new DeptForm();                  //创建 DeptForm 对象
        Enumeration<String> e=request.getParameterNames();      //获取参数
        try{                                           //遍历参数
            while(e.hasMoreElements())
            {
                String fname=e.nextElement().toString();
                String value=request.getParameter(fname);
                BeanUtils.setProperty(form,fname,value);
            }
        }catch(Exception ex){
          ex.printStackTrace();
        }
        this.getService().save(form);                  //保存 form 对象
        response.sendRedirect("./deptnew.jsp");    //实现转向功能
    }
    //编写分页方法
    public  void  list(HttpServletRequest  request,  HttpServletResponse
    response)
        throws ServletException, IOException
    {
        int curpage=1;                                 //为 curpage 赋予初值
        String curpageStr=request.getParameter("curpage");
                                                       //获取传送过来的 curpage
        //判断 curpage 的值
        if(curpageStr!=null&&!curpageStr.equals(""))
            curpage=Integer.parseInt(curpageStr);
        String cond="";
        Pageinfo pageinfo=this.getService().getDepts(curpage, cond);
        request.setAttribute("pageinfo",pageinfo); //设置相应的 Session 对象
        //实现转向
        request.getRequestDispatcher("./dept.jsp").forward(request,
        response);
    }
}
```

接着在 web.xml 页面中配置该 Servlet 程序。

```
<!--配置 DeptServlet 类路径-->
<servlet>
  <servlet-name>DeptServlet</servlet-name>
  <servlet-class>org.cjg.actions.DeptServlet</servlet-class>
</servlet>
<!--配置 DeptServlet 映射路径-->
<servlet-mapping>
  <servlet-name>DeptServlet</servlet-name>
  <url-pattern>/dept</url-pattern>
</servlet-mapping>
```

【代码解析】

- 在编写 doPost()方法时，当请求为 save()方法时，则会调用该类中的 save()方法；当请求为 list()方法时，则会调用该类的 list()方法。
- 在编写 list()方法时，首先确定属性 curpage 的值，然后通过调用业务层的 getDepts()

方法来创建出 Pageinfo 对象，最后把该对象存储到 Session 对象中。

18.6.2　Hibernate 分页系统所涉及的页面

通过 DeptServlet.java 文件的内容可以发现，Hibernate 分页系统需要 4 个页面：welcome.jsp、dept.jsp、deptnew.jsp 和 exception.jsp。

1．欢迎页面

如果想查看分页效果，可以先打开欢迎界面，该界面的具体内容如代码 18.18 所示。当单击“部门信息”链接后，就会在名为 main 的框架中显示出相关内容。

代码 18.18　欢迎页面：welcome.jsp

```
...
  <body>
      <td >
        <!--链接 -->
        <li><a href="./dept?m=list" target="main">部门信息</a>
      </td>
      <td ">
        <!--框架-->
        <iframe src="main.jsp" border=0 frameborder=0 width="100%" height=
        "100%" name="main">
        </iframe>
      </td >
  </body>
...
```

注意： 在上述代码中，当单击“部门信息”链接后，就会在名为 mail 框架中显示出调用类 DeptServlet.list()处理的结果，mail 框架只是一个空页面。

2．显示部门信息

当运行完类 DeptServlet 中的 list()方法后，就会自动转向 dept.jsp 页面，该页面的具体内容如代码 18.19 所示。

代码 18.19　显示部门信息：dept.jsp

```
...
  <body>
   <!--某页的数据 总记录数 总页数 当前页 分页单位-->
   <table border="1" align="center" width="80%">
    <c:forEach items="${pageinfo.pagedata}" var="dept">
      <tr>
        <td>${dept.deptid}</td>                <!--部门 ID-->
        <td>${dept.deptname}</td>              <!--部门名称-->
        <td>${dept.deptnum}</td>               <!--部门编号-->
        <td>${dept.deptdesc}</td>              <!--部门描述-->
      </tr>
    </c:forEach>
   </table>
```

```
  <c:if test="${pageinfo.curpage>1}">               <!--设置首页和上一页-->
    <a href="./dept?m=list">首页</a>
    <a href="./dept?m=list&curpage=${pageinfo.previouspage}">上一页</a>
  </c:if>
  <c:if test="${pageinfo.curpage<pageinfo.allpage}"> <!--设置尾页和下一页
  -->
    <a href="./dept?m=list&curpage=${pageinfo.nextpage}">下一页</a>
    <a href="./dept?m=list&curpage=${pageinfo.allpage}">尾页</a>
  </c:if>
 </body>
...
```

3．添加记录页面

当想添加关于部门 dept 的记录时，只要在 deptnew.jsp 页面中填写相应的信息，单击“提交”按钮就可以实现记录的添加。该页面的具体内容如代码 18.20 所示。

代码 18.20　添加记录页面：deptnew.jsp

```
...
 <body>
  <form action="./dept?m=save" method="post">
...
        <!--部门名称输入框-->
        <td>部门名称</td><td><input name="deptname"></td>
        <!--部门编制输入框-->
        <td>部门编制</td><td><input name="deptnum"></td>
        <!--部门描述输入框-->
        <td>部门名称</td><td><textarea name="deptdesc" cols="60"
        rows="2"></textarea>
        <!--提交按钮-->
        <input type="submit" value="提交">
...
  </form>
 </body>
...
```

注意： 在上述代码中，当单击“提交”按钮后，就会由类 DeptServlet 中的 save()方法来处理。

18.7　多学两招——分页标签

当显示给浏览者的内容很多时，如果不想一页全部显示出来，而是分成若干多个页来显示，就是分页显示。分页显示在开发过程中经常会遇到，几乎每一个 Web 应用都会涉及。当在开发分页显示时，开发人员可以自己开发底层代码，如 Hibernate 分页系统，但是更方便的方法，可以使用一些现有的技术，如分页标签 Displaytag 和标签 Pager。

18.7.1　下载 Displaytag 分页标记库

在 Java Web 项目中，使用 Displaytag 标记库可以非常方便地实现表格方式的分页显示。

Displaytag 标记库的最新版本为 1.2，具体的下载步骤如下。

（1）首先访问下载 Displaytag 标记库的官方网站（http://displaytag.sourceforge.net），如图 18.19 所示。

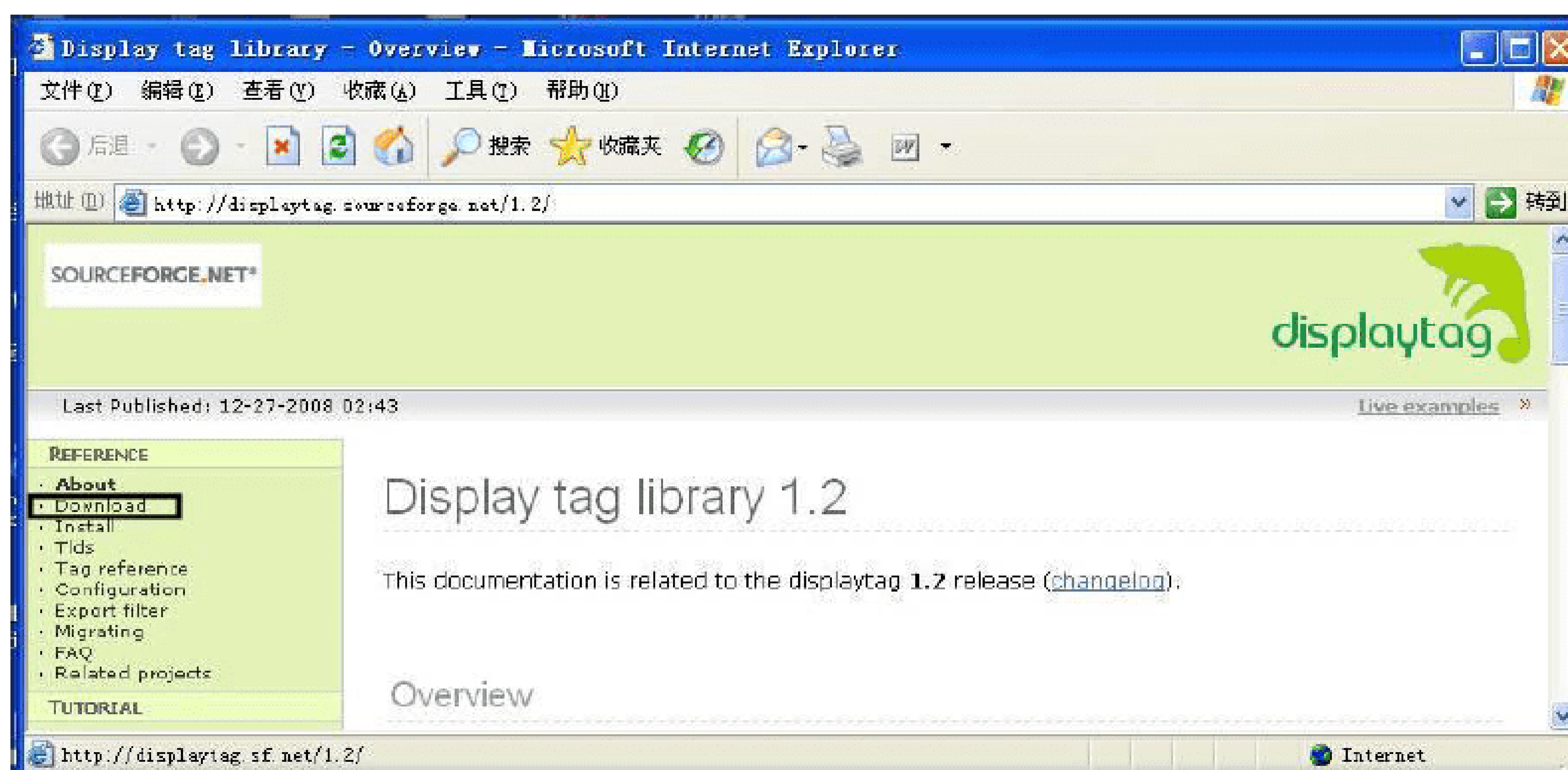


图 18.19　Displaytag 标记库首页

（2）打开 Displaytag 标记库的官方网站首页后，在其左边的 REFERENCE 项目中单击 Download 链接后，就可以打开关于下载 Displaytag 标记库的页面，如图 18.20 所示。

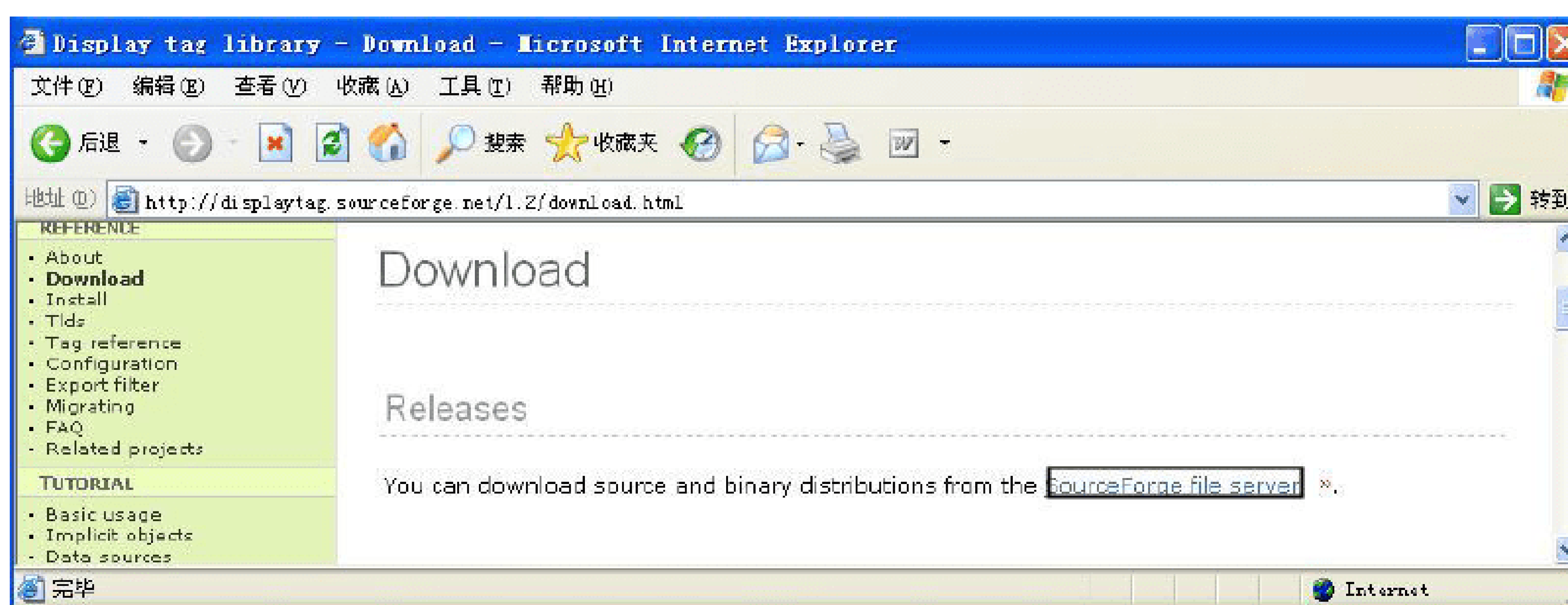


图 18.20　关于下载 Displaytag 标记库的页面

（3）在上述页面中单击 SourceForge file server 链接，就可以转到下载 Displaytag 标记库的页面，如图 18.21 所示。在该页中单击 Download 链接后就可以转到选择下载类型的页面。

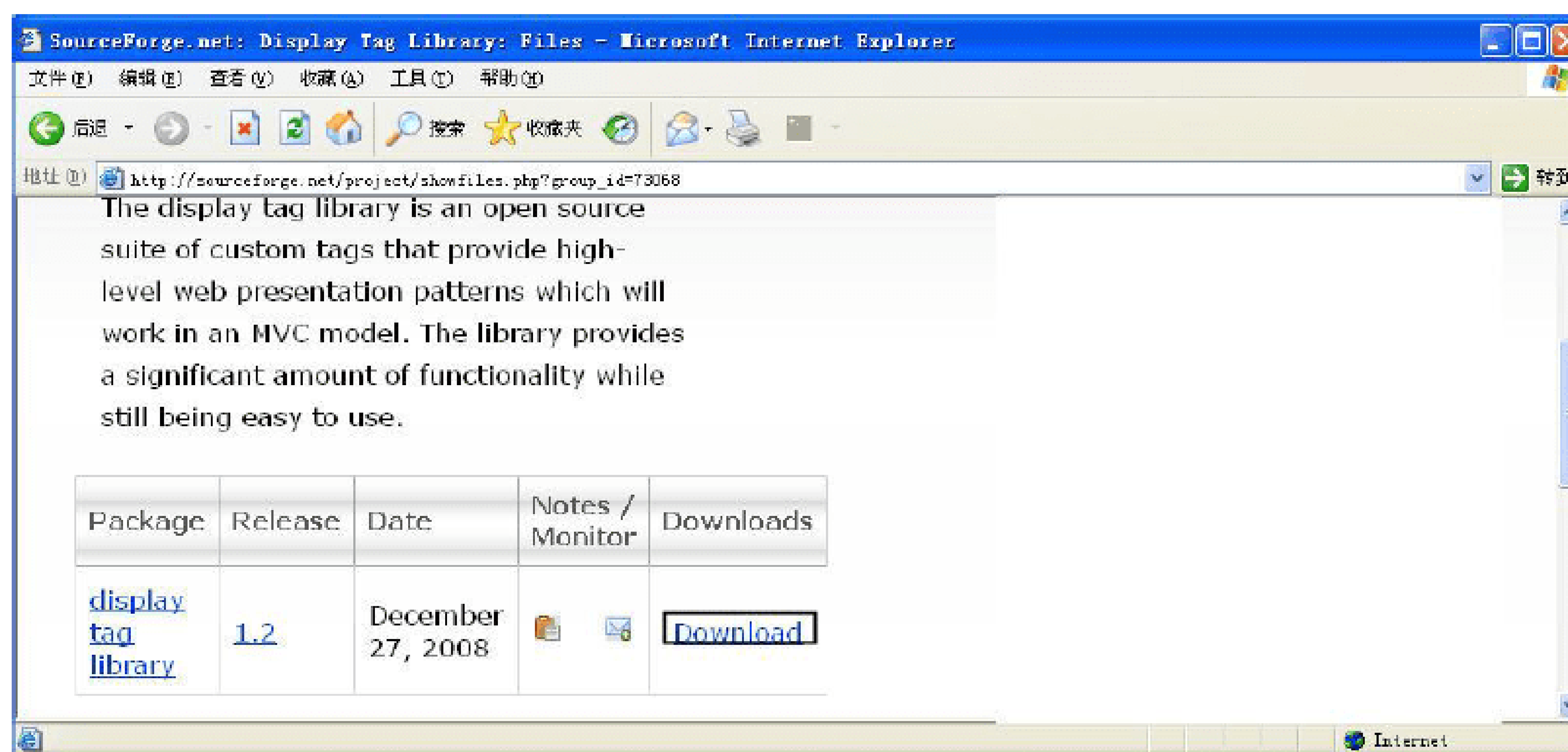


图 18.21　下载页面

（4）在选择下载 Displaytag 标记库类型的页面中，因为不需要源代码，所以只下载 displaytag-1.2-bin.zip 就可以。单击 displaytag-1.2-bin.zip 链接就可以实现该 jar 文件的下载，如图 18.22 所示。

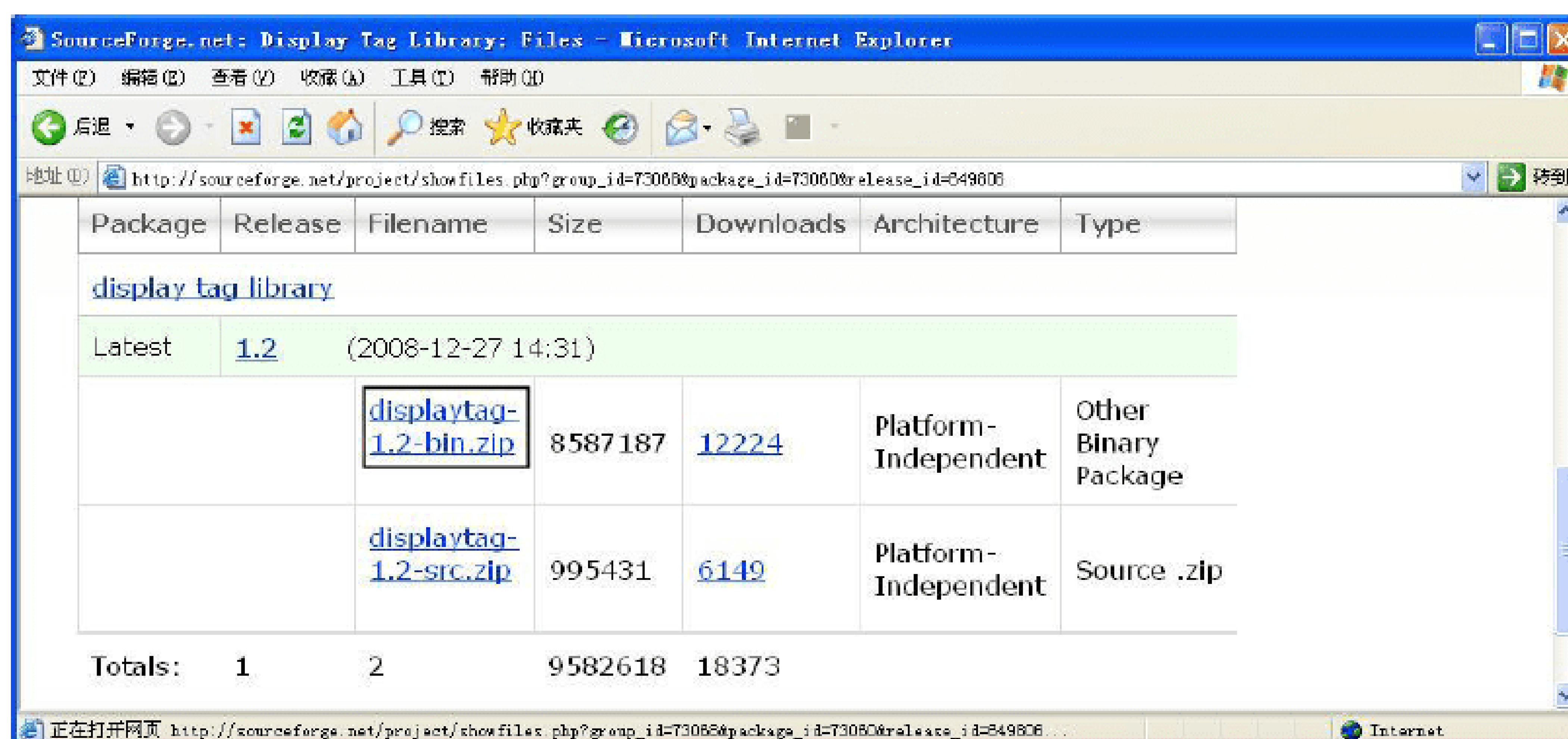


图 18.22 下载 Displaytag 标记库

至此，就完成对 Displaytag 标记库的下载。

18.7.2 了解和配置 Displaytag 标记库

18.7.1 节介绍了如何下载 Displaytag 标记库，下载完该标记库后就可以在 Java Web 项目中使用该框架。在具体使用之前，先解压 displaytag-1.2-bin.zip 文件，该压缩包目录如图 18.23 所示，各个文件的作用如下。

名称	大小
..	
docs	
displaytag-portlet-1.2.jar	8,785
displaytag-export-poi-1.2.jar	12,758
displaytag-1.2.jar	219,121
LICENSE.txt	4,978
displaytag-examples-1.2.war	4,342,677

图 18.23 目录结构

- docs：Displaytag 标记库的帮助文档。
- displaytag-1.2.jar：Displaytag 标记库的核心类库。
- displaytag-examples-1.2.war：关于 Displaytag 标记库的实例。
- displaytag-export-poi-1.2.jar 和 displaytag-portlet-1.2.jar：关于 Displaytag 标记库实现导出功能的相关 jar。

由于 Displaytag 标记库在某些方面存在缺陷，所以经常会使用到一些辅助的 jar 文件，对于这些 jar 文件可以从相关网站下载，也可以从 displaytag-examples-1.2.war 实例架包中获取。为了方便使用，可以复制 displaytag-examples-1.2.war\WEB-INF\lib 文件下的所有 jar 文件到相应的文件夹里，然后为这些辅助 jar 文件与 Displaytag 标记库的核心 jar 文件，在 MyEclipse 开发环境中专门建立一个名为 displaytag 的用户库，如图 18.24 所示。

至此，就完成了关于 Displaytag 标记库的用户库配置。

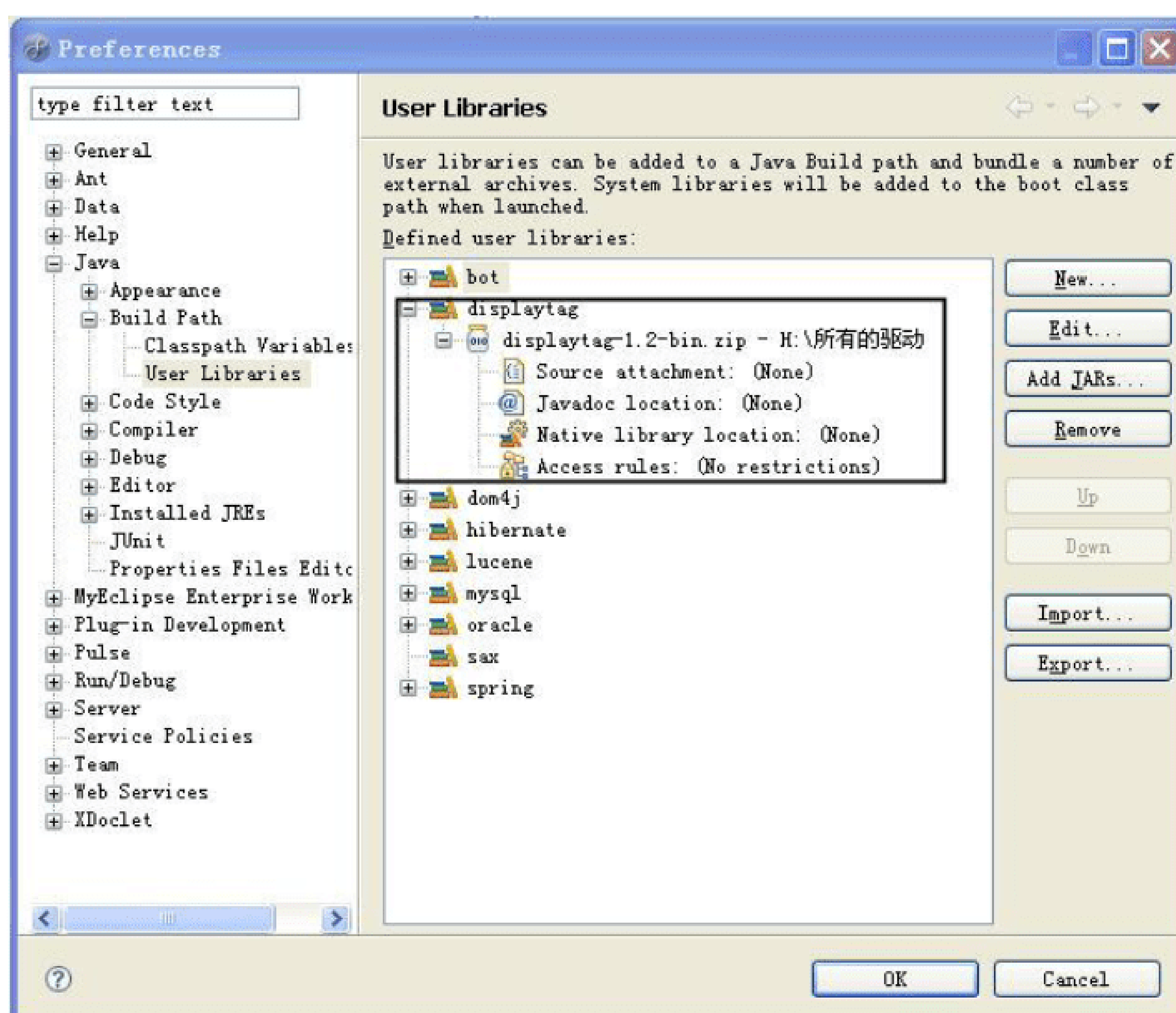
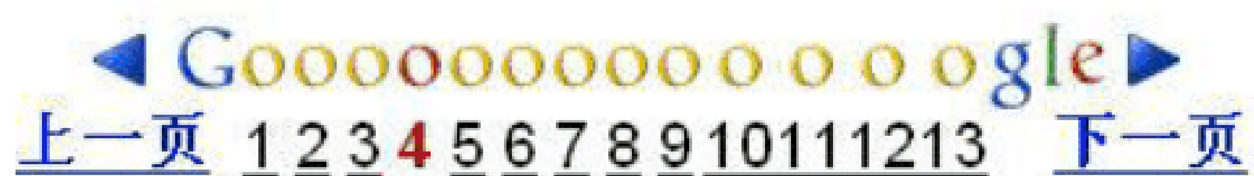


图 18.24　displaytag 用户库

18.7.3　下载和配置 Pager 标记库

Pager 标记库的历史很悠久，在很早以前就出现了。可以利用十分灵活的方式实现各种分页显示效果，例如模仿 Google 的分页效果，如图 18.25 所示。Displaytag 标记库的最新版本为 2.0，具体的下载步骤如下。

（1）首先访问下载 Pager 标记库的官方网站（http://jsptags.com），如图 18.26 所示。



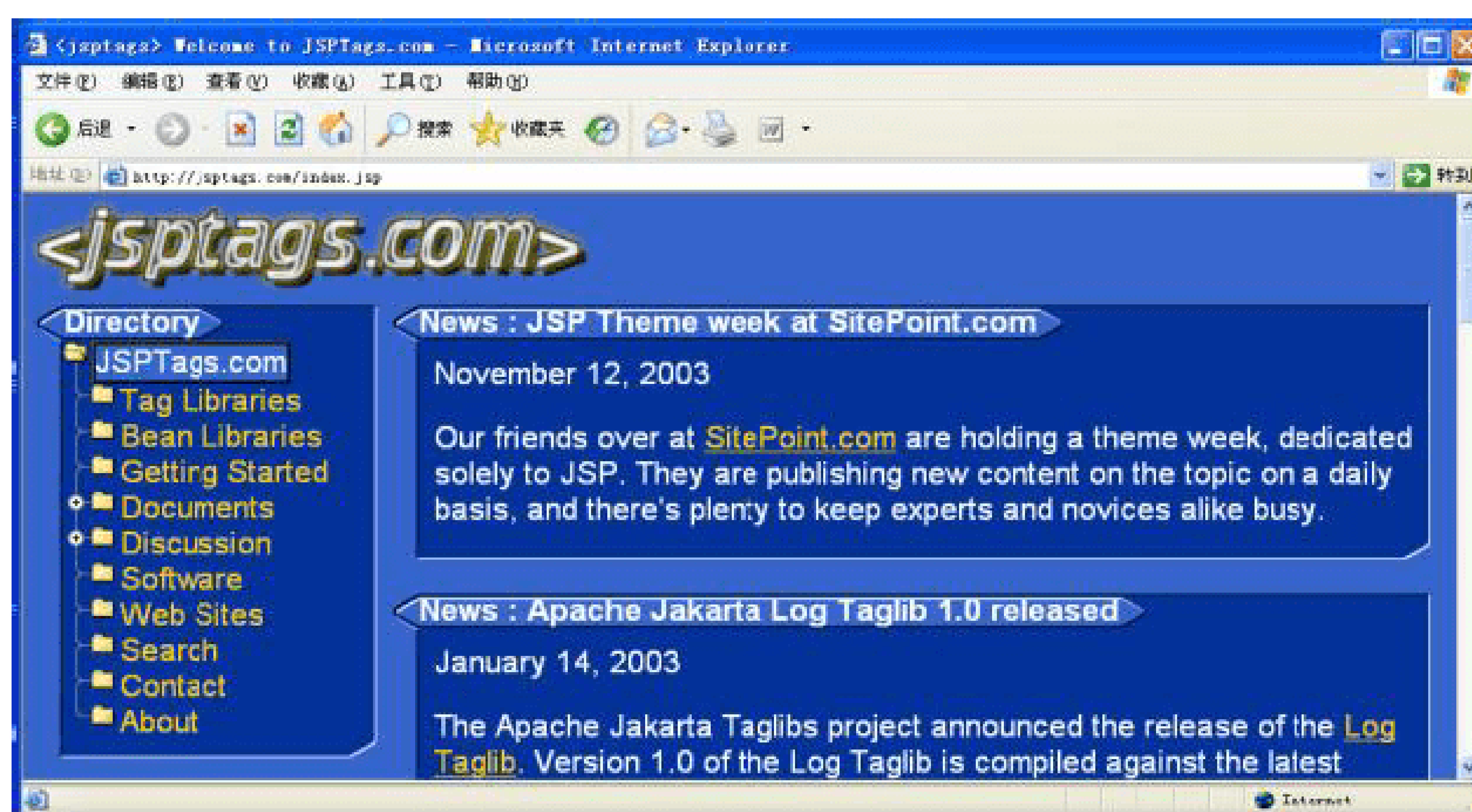


图 18.25　模仿 Google 分页效果　　　　图 18.26　Pager 标记库首页

（2）打开 Pager 标记库的官方网站首页后，在其左边的 Directory 目录中单击 Tag Libraries 链接后，就可以打开关于标记库的页面，如图 18.27 所示。接着在该页面中单击 JSPTags.com Pager Tag Library 链接，就可以进入下载关于 Pager 标记库的页面。

（3）在关于 Pager 标记库的页面（如图 18.28 所示）中：Demo 表示关于 Pager 标记库的演示实例；Documentation 表示关于 Pager 标记库的开发文档和 Download 表示关于 Pager 标记库的下载页面。

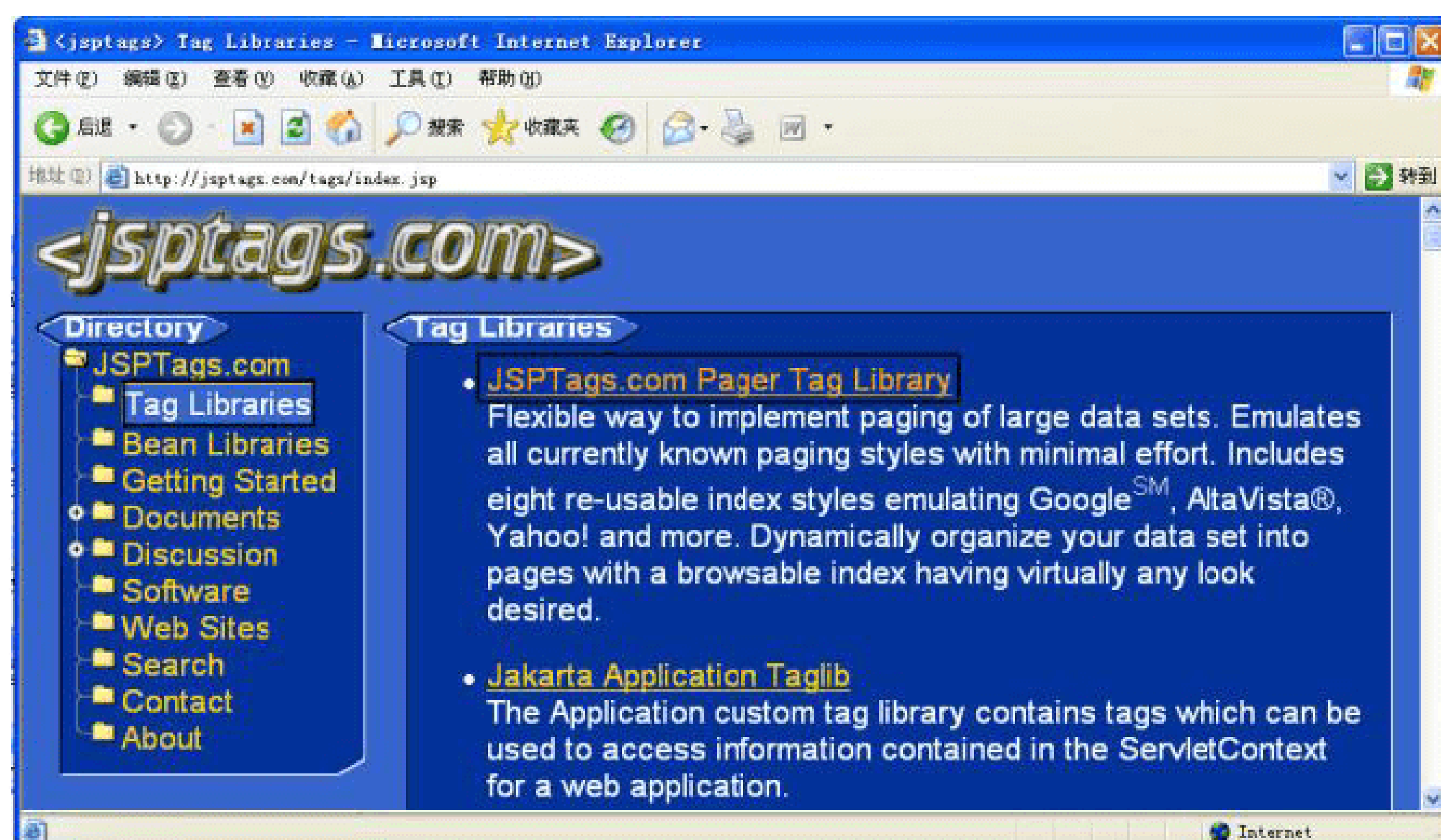


图 18.27　关于下载 Displaytag 标记库的页面



图 18.28　选择 Pager 标记库相关类型

（4）单击 Download 链接就可以进入下载 Pager 标记库的真正页面（如图 18.29 所示），在该页面中单击 pager-taglib-2.0.war 链接就可以下载该标记库。

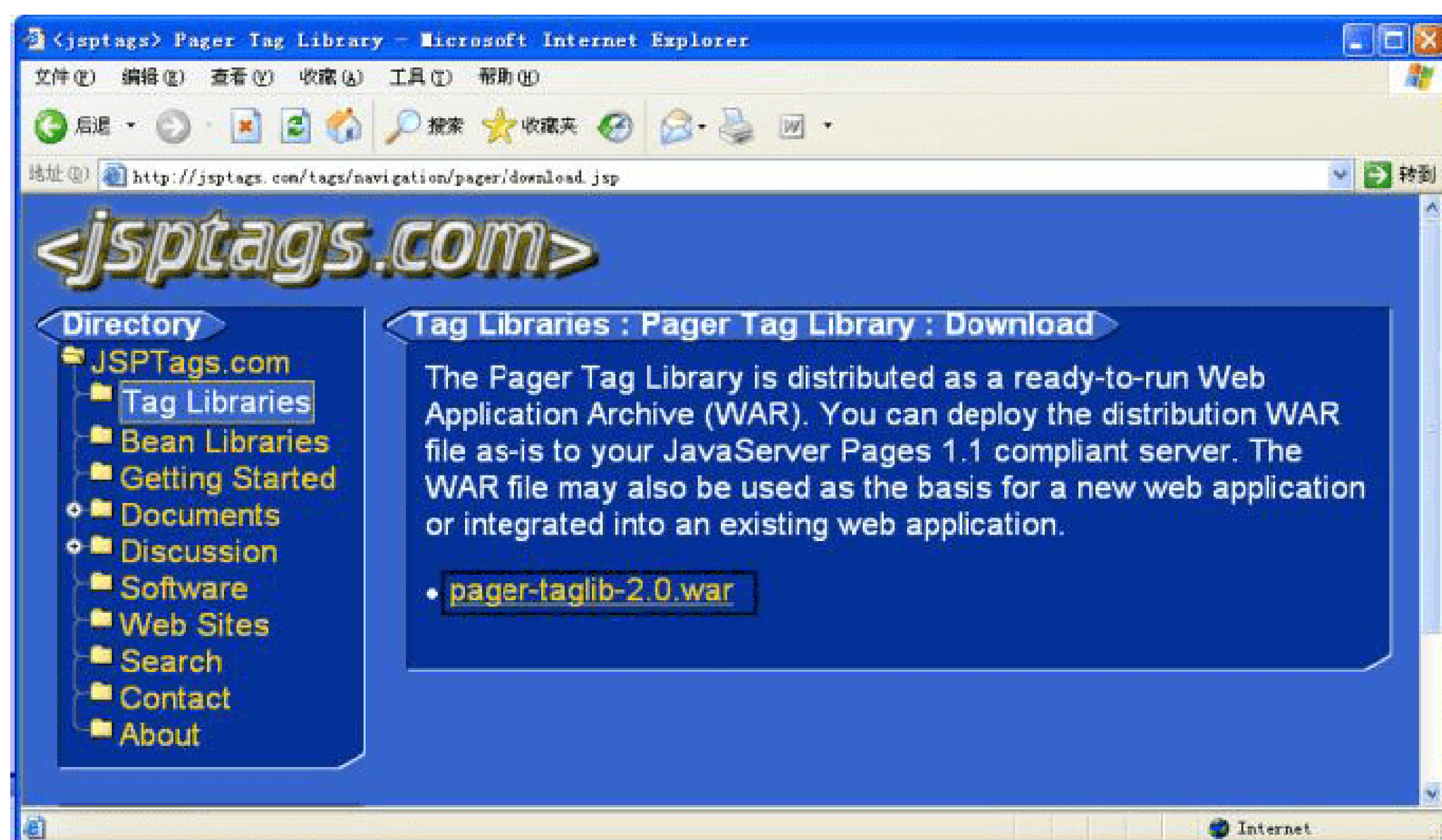


图 18.29　下载 Pager 标记库的真正页面

至此，就完成对 Pager 标记库的下载。下载完该标记库后就可以在 Java Web 项目中使用该标记。在具体使用之前，先解压 pager-taglib-2.0.war 文件，pager-taglib-2.0.war\WEB-INF\lib 目录中的文件如图 18.30 所示，各个文件的作用如下。

❑ pager-taglib.jar：pager 标记库的核心类库。

名称	大小	压缩后大小
..		
pager-taglib.jar	46,875	42,498
pager-src.jar	43,767	39,355

图 18.30　目录结构

❑ pager-src.jar：pager 标记类库的源代码。

为了使用方便，在 MyEclipse 开发环境中专门建立一个名为 pager 的用户库，如图 18.31 所示。

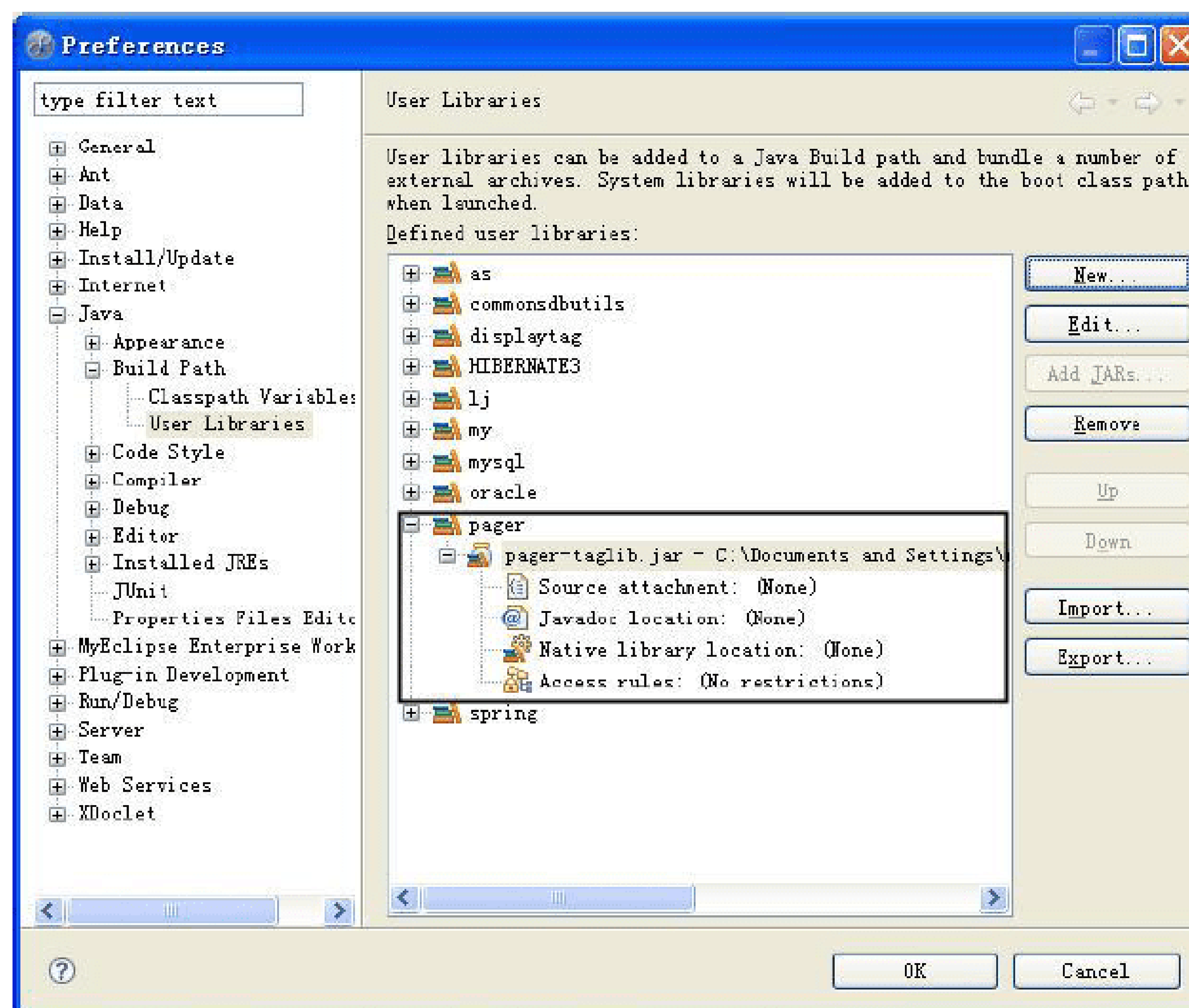


图 18.31　pager 用户库

18.8　小　　结

本章主要介绍了 Hibernate 分页系统，本系统基于 Hibernate 3.0 框架构建而成。为了让读者深入理解 Hibernate 分页系统，首先讲解了封装 JavaBean 操作的 Commons-BeanUtils 组件，接着为了使用方便，封装了关于 Hibernate 框架中各种对象的操作。最后，还详细讲解了关于分页的 Displaytag 标签和 Pager 标签。在具体实现 Hibernate 分页系统时，不仅通过单纯的 Hibernate 3.0 框架实现，而且具体的分页功能还是通过调用程序员自己编写的方法来实现。

第 19 章　生成报表（Struts 2.x+Hibernate+JXL）

一个功能强大的网站系统经常会遇到这种情形，浏览者通过 IE 浏览器请求 Web 服务器里的数据，而查询结果则是上千条甚至是上万条记录。对于这些记录如何处理才能以 Microsoft Excel 工作表的形式保存到浏览者自己的计算机中，这就需要一种非常实用的技术——报表技术。本章将详细地介绍如何在 Struts 2.0+Hibernate+JXL 框架中实现生成报表。

19.1　生成报表原理

在 Java Web 项目中可以通过各种方式实现报表模块功能，例如利用 Jakarta POI 组件实现报表功能、利用 JasperReport 组件实现报表功能等。本章将通过在 Struts 2.0+Hibernate 框架中调用 JXL（Java Excel API）组件来实现报表功能。

19.1.1　生成报表框架分析

对于一个大型网站系统来说，实现一个可用的报表功能要考虑的情况十分复杂，例如，如何合理规划各种报表的格式和如何尽快地生成报表等。本章将会实现一个比较简单可用的报表模块，读者可以根据自己的需求进行完善。本系统项目目录如图 19.1 所示。

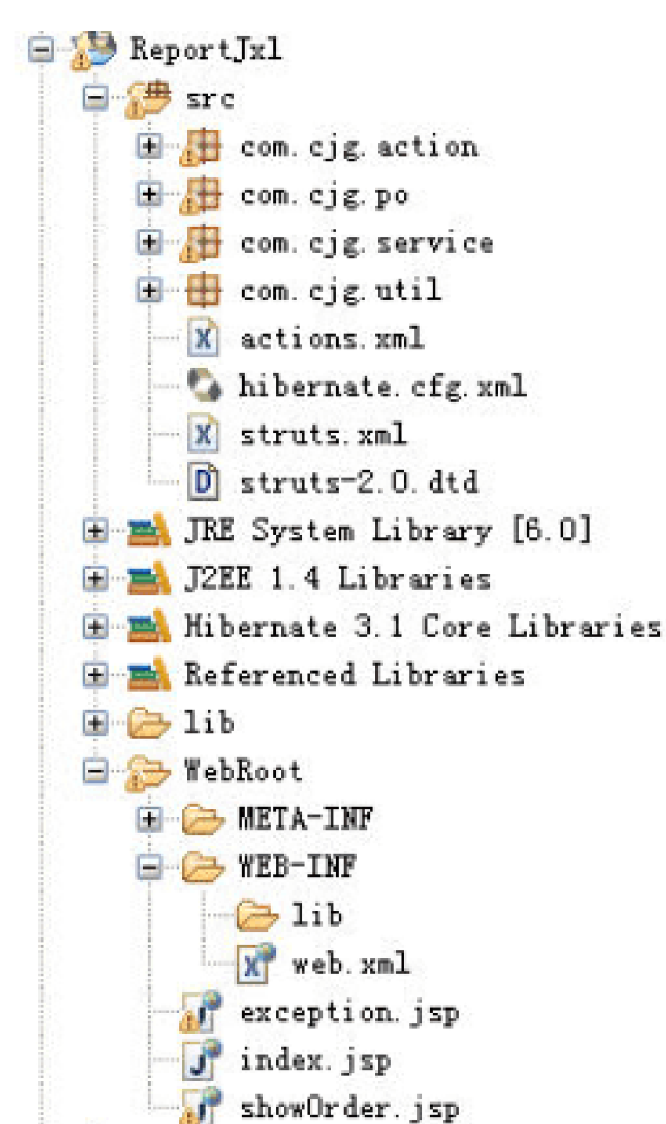


图 19.1　项目目录

19.1.2　生成报表功能描述

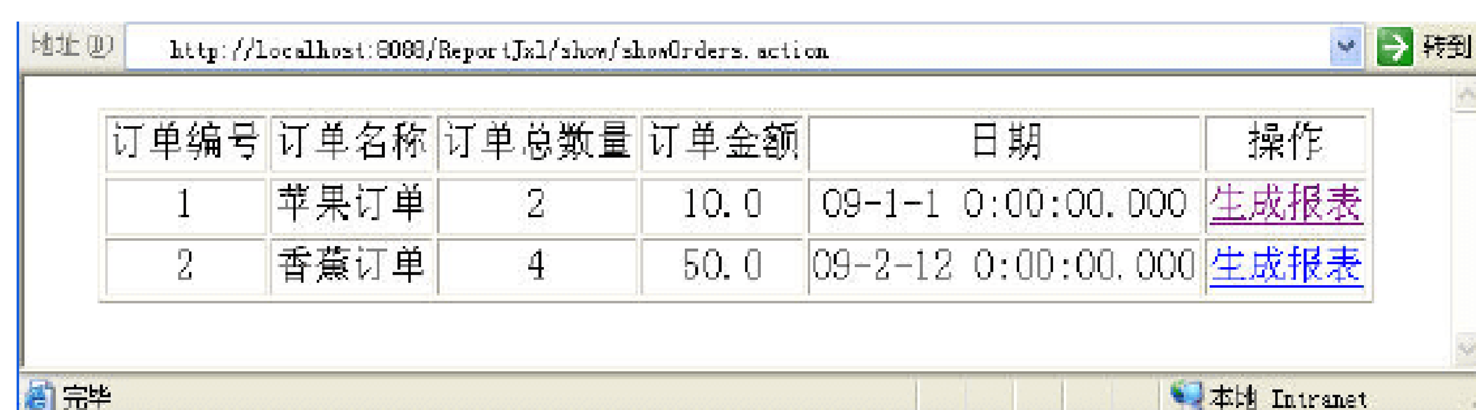
本节将以直观的方式来向读者介绍整个生成报表模块要实现的功能。为了方便讲解，将通过生成订单信息和订单明细报表的具体实例来介绍生成报表模块的各个功能。这些功能包括展示订单信息和生成相应订单信息、订单明细报表。

1．展示订单信息

首先通过访问地址 http://localhost:8088/ReportJxl/index.jsp 来打开展现所有订单的页面，该页面如图 19.2 所示。在该页面中单击“展现所有订单”链接就可以转到显示所有订单信息的页面，如图 19.3 所示。



图 19.2　展示订单页面



订单编号	订单名称	订单总数量	订单金额	日期	操作
1	苹果订单	2	10.0	09-1-1 0:00:00.000	生成报表
2	香蕉订单	4	50.0	09-2-12 0:00:00.000	生成报表

图 19.3　展示订单信息页面

2. 生成和查看报表

当单击图 19.3 中两个订单记录中的“生成报表”链接后，就可以在服务器根目录的 report 文件中生成相应的报表，打开目录 Tomcat 的安装目录\webapps\ReportJxl\report，就可以发现如图 19.4 所示的文件。

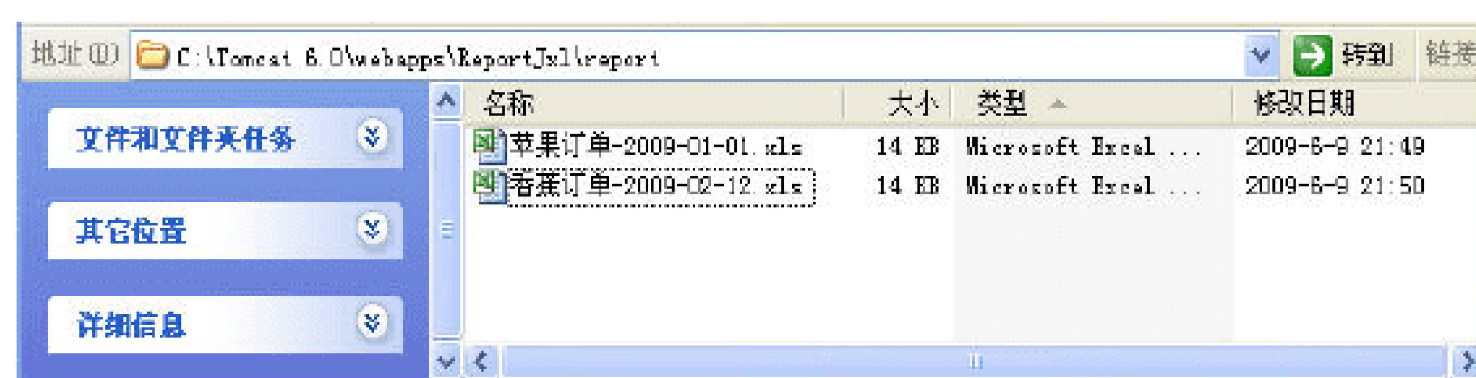


图 19.4　生成的报表

双击“苹果订单-2009-01-01.xls”文件，该文件的内容如图 19.5 所示。双击“香蕉订单-2009-02-12.xls”文件，该文件的内容如图 19.6 所示。

01月01日订单报表

订单编号	1		
订单名称	苹果订单		
订单总数量	2		
订单总价	10.0		
订单日期	2009-01-01		
打印日期	2009-06-09		
明细编号	**明细名称**	**数量**	**单价**
1	苹果明细	2	5.0

图 19.5　苹果订单-2009-01-01.xls 文件内容

02月12日订单报表

订单编号	2		
订单名称	香蕉订单		
订单总数量	4		
订单总价	50.0		
订单日期	2009-02-12		
打印日期	2009-06-09		
明细编号	**明细名称**	**数量**	**单价**
2	香蕉订单	5	4.0

图 19.6　香蕉订单-2009-02-12.xls 文件内容

19.2　下载 JXL 组件

JXL 全称 Java Excel API，通过 Java API 操作 Excel 文件的类库。JXL 类库不仅支持 Excel 175-2000 的所有版本，而且还不依赖 Windows 系统。即在 Linux 环境下，该类库依然能够正确地处理 Excel 文件。

19.2.1　下载 JXL 组件

到目前为止，在 Java Web 项目中如果要实现生成报表，一般使用 Apache 组织的 POI 组件或 JXL 组件。由于 MyEclipse 开发环境中报表的插件为 JXL，所以在本章将使用 JXL 组件。目前 JXL 组件的最新版本为 2.6.10，具体下载步骤如下。

（1）首先访问下载 JXL 报表相关 jar 文件的官方网站（http://www.JexcelApi.sourceforge.net），如图 19.7 所示。在该页面中单击 Resources 目录下的 download 链接就可以转到 JXL 组件的相关页面。

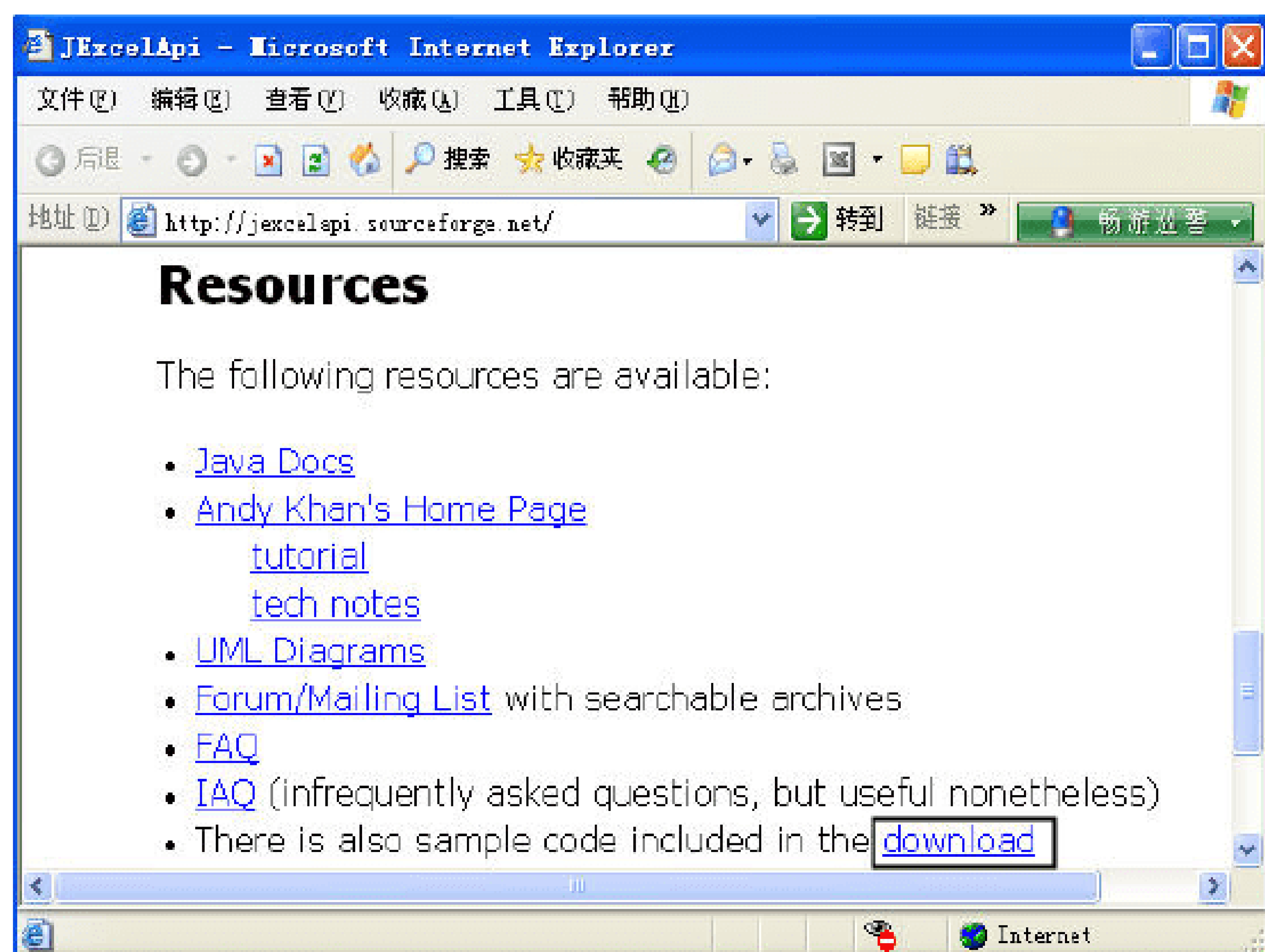


图 19.7　JXL 组件首页

（2）在 JXL 组件相关页面中（如图 19.8 所示），单击 jexcelapi_2_6_12.zip 链接就可以实现该组件的下载。

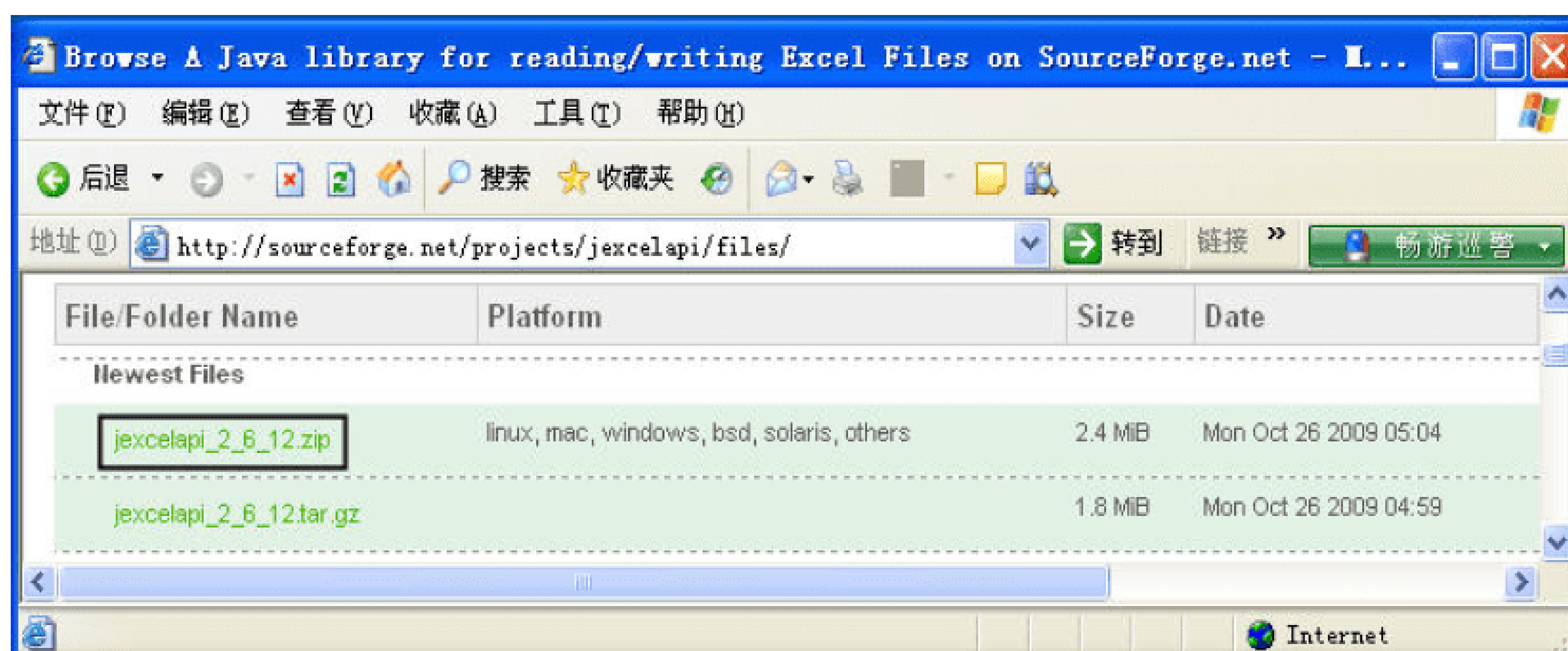


图 19.8　JXL 组件的下载

至此，就完成了对 JXL 组件的下载。

19.2.2　安装和配置 JXL 报表组件

19.2.1 节介绍了如何下载 JExcelApi 报表，下载完该报表的 jar 后就可以在 Java Web 项目中使用该框架。在具体使用之前，先解压 jexcelapi_2_6_10.zip 文件，该压缩包目录如

图 19.9 所示。各个文件的作用如下。

- ❑ src：JexcelApi 相关 jar 包的源代码；
- ❑ docs：JexcelApi 相关 jar 包的帮助文档；
- ❑ jxl.jar：JexcelApi 相关 jar 包。

为了便于使用，可以复制 jxl.jar 文件到相应的文件夹中，然后为这个文件在 MyEclipse 开发环境中专门建立一个名为 jxl 用户库，如图 19.10 所示。

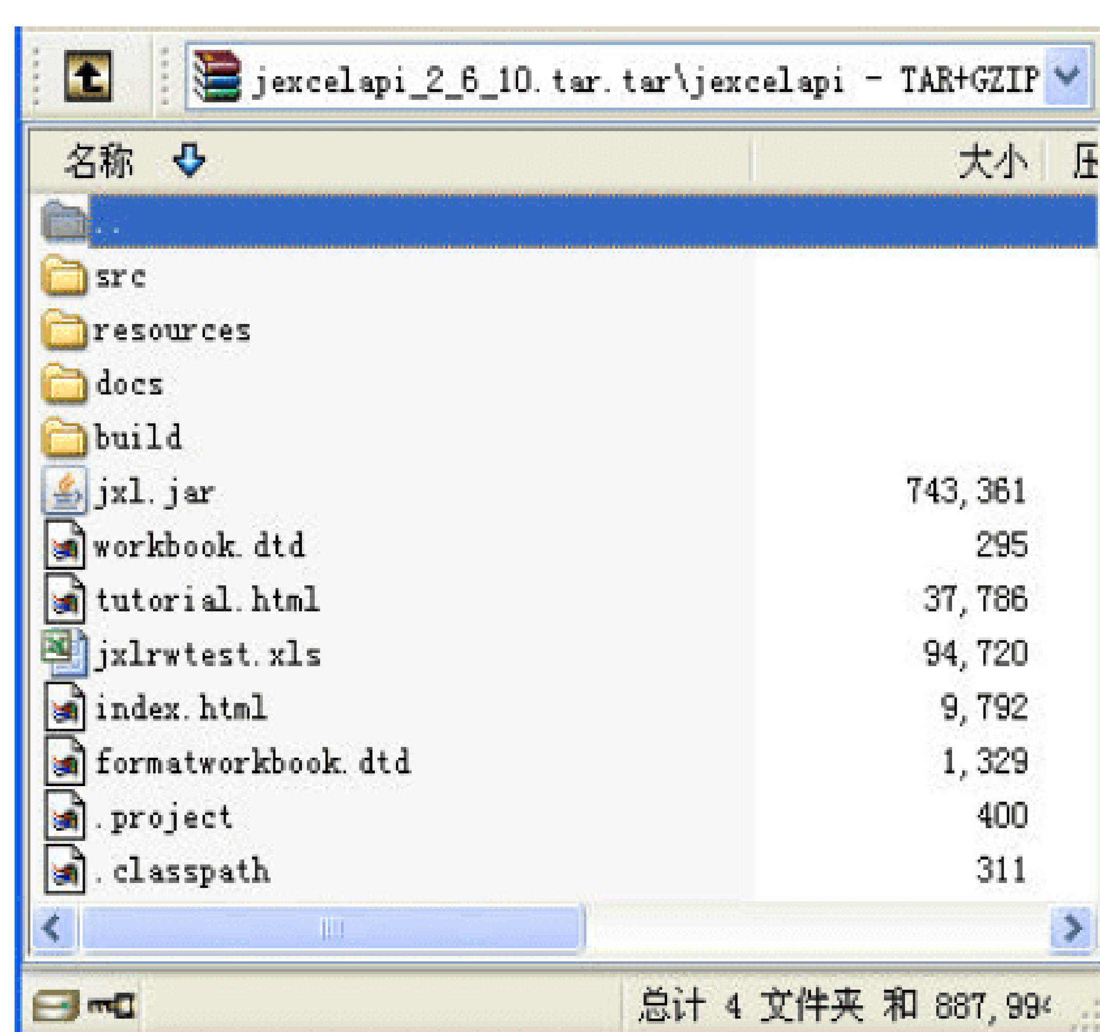


图 19.9　目录结构

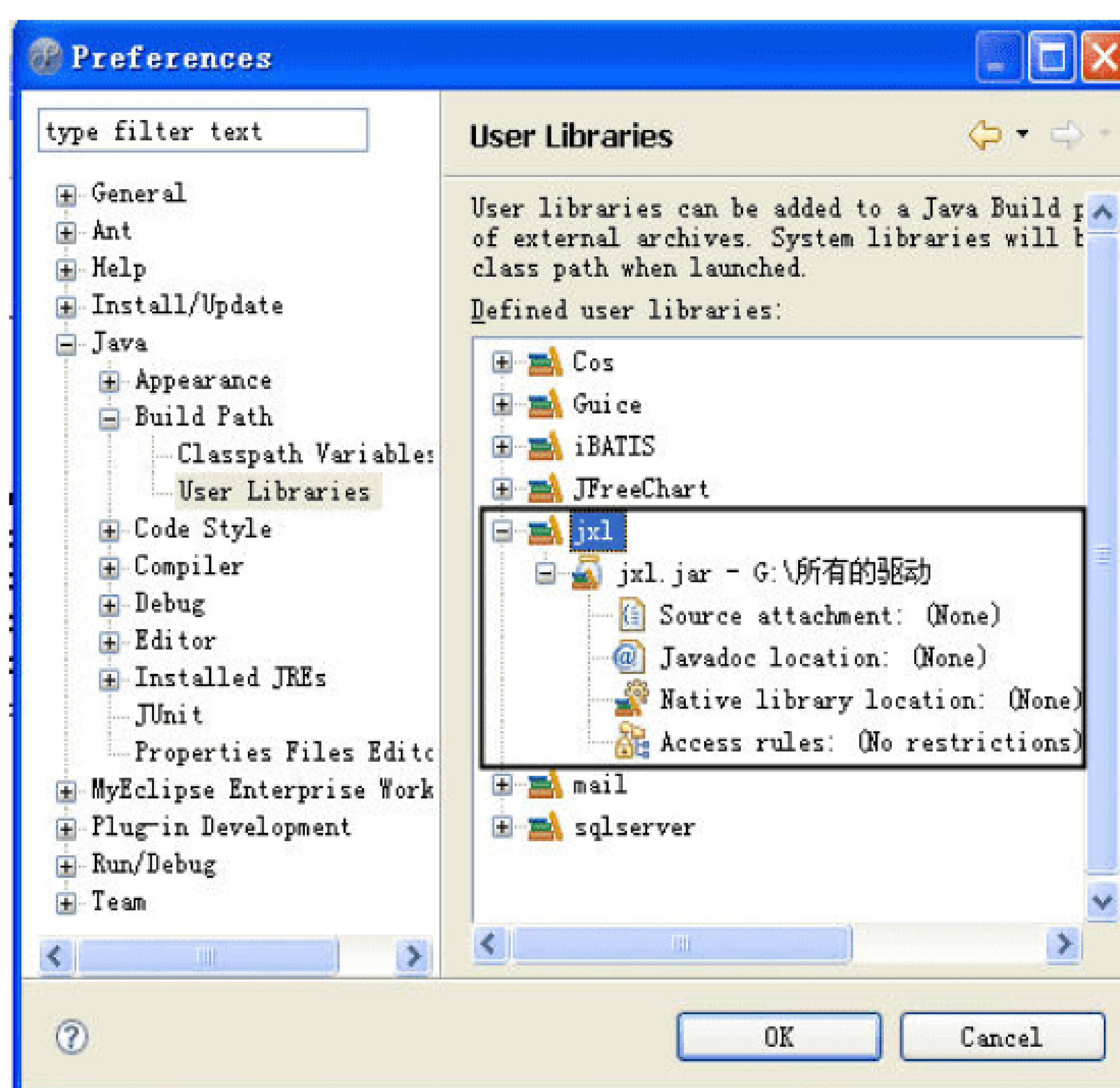


图 19.10　配置 jxl 用户库

至此，就完成了 jxl 用户库的配置。

19.3　生成报表前期准备

本节除了将详细的介绍如何设计关于生成报表功能的数据库和表外，还将配置实现该系统将利用的 Struts 2.x+Hibernate+JXL+MySQL 框架的环境。其中 Struts 2.x 框架的版本号为 Struts 2.0、Hibernate 框架的版本号为 Hibernate 3.0、数据库 MySQL 的版本号为 MySQL 5.0。

19.3.1　数据库设计

生成报表功能需要建立一个数据库并在该数据库中建立两张表，即存放表格的数据库 jxl、存放订单信息的表 orderinfo 和存放订单明细的表 orderitem。

1. 创建数据库jxl

如果想创建出数据库 jxl，可以在 MySQL 的命令行窗口中输入如下命令：

```
CREATE DATABASE 'jxl'
```

2．创建表orderinfo

如果想创建出表 orderinfo，可以在 MySQL 的命令行窗口中输入相关命令。该表格的模拟数据如图 19.11 所示。

orderid	ordermc	totalnum	totalje	orderinfodate
1	苹果订单	2	10	2009-01-01 00:00:00
2	香蕉订单	4	50	2009-02-02 00:00:00

图 19.11　orderinfo 表模拟数据

该表的具体信息如表 19.1 所示。

表 19.1　表orderinfo信息

字段名称	数据类型	字段说明	字段名称	数据类型	字段说明
orderid	int(11)	订单编号	totalje	double	订单总价
ordermc	varchar(30)	订单名称	orderinfodate	datetime	订单时间
totalnum	int(11)	订单总数量			

实现订单信息模型的内容如代码 19.1 所示。关于该类的映射文件内容如代码 19.2 所示。

代码 19.1　Orderinfo 对象：Orderinfo.java

```
...
public class Orderinfo  implements java.io.Serializable {
    //创建各种字段
    private Integer orderid;                            //订单 ID 变量
    private String ordermc;                             //订单名称变量
    private Integer totalnum;                           //订单总数量变量
    private Double totalje;                             //订单总价变量
    private Date orderinfodate;                         //订单时间变量
    private Set orderitems = new HashSet(0);
    public Orderinfo() {                                //无参构造函数
    }
    public Orderinfo(Integer orderid) {                 //有参构造函数
      this.orderid = orderid;
    }
    //有参构造函数
    public Orderinfo(Integer orderid, String ordermc, Integer totalnum,
    Double totalje, Date orderinfodate, Set orderitems) {
      this.orderid = orderid;
      this.ordermc = ordermc;
      this.totalnum = totalnum;
      this.totalje = totalje;
      this.orderinfodate = orderinfodate;
      this.orderitems = orderitems;
}
//省略 orderid、ordermc、totalnum、totalje、orderinfodate 字段属性配置

}
```

代码 19.2　Orderinfo 对象映射文件：Orderinfo.hbm.xml

```
...
<hibernate-mapping>
<class name="com.cjg.po.Orderinfo" table="orderinfo" catalog="Jxl">
      <!--制定要持久化的类与对应数据库的表映射关系-->
```

```
        <id name="orderid" type="java.lang.Integer">
            <column name="orderid" />
            <generator class="assigned" />
        </id>
        <!--表 orderinfo 字段 ordermc 与 ordermc 属性的映射关系-->
        <property name="ordermc" type="java.lang.String">
            <column name="ordermc" length="30" />
        </property>
        <!--表 orderinfo 字段 totalnum 与 totalnum 属性的映射关系-->
        <property name="totalnum" type="java.lang.Integer">
            <column name="totalnum" />
        </property>
        <!--表 orderinfo 字段 totalje 与 totalje 属性的映射关系-->
        <property name="totalje" type="java.lang.Double">
            <column name="totalje" precision="8" />
        </property>
        <!--表 orderinfo 字段 orderinfodate 与 orderinfodate 属性的映射关系-->
        <property name="orderinfodate" type="java.util.Date">
            <column name="orderinfodate" length="10" />
        </property>
        <set name="orderitems" inverse="true">
            <key>
                <column name="orderid" />
            </key>
            <one-to-many class="com.cjg.po.Orderitem" /> <!--实现一对多映射-->
        </set>
    </class>
</hibernate-mapping>
```

注意： Orderinfo.java 类可以参考数据库中的表格 orderinfo 来编写，而该类的映射文件则可以通过向导来实现。

3．创建表orderitem

如果想创建出表 orderitem，可以在 MySQL 的命令行窗口中输入相关命令。该表的具体信息如表 19.2 所示。

表 19.2　表orderitem信息

字段名称	数据类型	字段说明	字段名称	数据类型	字段说明
itemid	int(11)	明细编号	price	double	单价
orderid	int(11)	订单编号	num	int(11)	总数
productmc	varchar(30)	数量			

该表的模拟数据如图 19.12 所示。

	itemid	orderid	productmc	price	num
	1	1	苹果明细	5.0	2
	2	2	香蕉订单	4.0	5

图 19.12　orderitem 表模拟数据

实现订单明细模型的内容如代码 19.3 所示。关于该类的映射文件内容如代码 19.4 所示。

代码 19.3　Orderitem 对象：Orderitem.java

```
...
public class Orderitem  implements java.io.Serializable {
    //创建字段
     private Integer itemid;                              //明细 ID 变量
     private Orderinfo orderinfo;                         //订单信息
```

```
    private String productmc;                    //数量变量
    private Double price;                        //单价变量
    private Integer num;                         //总数变量
   //构造函数
   public Orderitem() {                          //无参构造函数
   }
   public Orderitem(Integer itemid) {            //有参构造函数
       this.itemid = itemid;
}
   //有参构造函数
   public Orderitem(Integer itemid, Orderinfo orderinfo, String productmc,
   Double price, Integer num) {
       this.itemid = itemid;
       this.orderinfo = orderinfo;
       this.productmc = productmc;
       this.price = price;
       this.num = num;
}
   //省略 itemid、orderinfo、productmc、price 和 num 字段省略配置

}
```

代码 19.4　Orderitem 对象映射文件：Orderitem.hbm.xml

```
...
<hibernate-mapping>
<class name="com.cjg.po.Orderitem" table="orderitem" catalog="Jxl">
       <!--制定要持久化的类与对应数据库的表映射关系-->
       <id name="itemid" type="java.lang.Integer">
           <column name="itemid" />
           <generator class="assigned" />
       </id>
       <!--表 orderitem 字段 orderid 与属性 orderinfo 的映射关系>
       <many-to-one name="orderinfo" class="com.cjg.po.Orderinfo" fetch=
      "select">
           <column name="orderid" />
       </many-to-one>
       <!--表 orderitem 字段 price 与属性 price 的映射关系>
       <property name="productmc" type="java.lang.String">
           <column name=" price " length="30" />
       </property>
       <!--表 orderitem 字段 price 与属性 price 的映射关系>
       <property name="price" type="java.lang.Double">
           <column name="price" precision="8" />
       </property>
       <!--表 orderitem 字段 num 与属性 num 的映射关系>
       <property name="num" type="java.lang.Integer">
           <column name="num" />
       </property>
   </class>
</hibernate-mapping>
```

注意：Orderitem.java 类可以参考数据库中的表格 orderitem 来编写，而该类的映射文件则可以通过向导来实现。

至此，就完成了对生成报表功能数据库和表的设计。

19.3.2　关于 Struts 2.x 的准备

在实现 Struts 2.x 框架与 Hibernate 3.0 框架二者集成时，对于其中的 Struts 2.x 框架除了需要引入相应的 jar 包外，还必须得对 struts.xml 和 web.xml 文件做相应的配置。

1. 引入jar文件

首先引入关于 Struts 2.0 框架的核心包：struts2-core-2.0.11.jar、xwork-2.0.4.jar、ognl-2.6.11.jar、freemarker-2.3.8.jar 和 commons-logging-1.0.4.jar。由于需要连接数据库 MySQL，所以还需要引入关于该数据库的驱动 mysql-connector-java-3.1.14-bin.jar。

注意：前 6 个 jar 文件在 Struts 2.0 框架的 jar 文件中就可以找到，而最后一个关于数据库的驱动则必须从该数据库的官方网站上下载。

2. 修改web.xml文件

为了使生成的报表项目支持 Struts 2.0 框架，需要在 web.xml 文件中增加如代码 19.5 所示的内容。

代码 19.5　修改 web.xml 文件：web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app 2 4.xsd">
    <!--设置过滤器类-->
    <filter>
        <filter-name>s</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <!--设置过滤器类映射-->
    <filter-mapping>
        <filter-name>s</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

3. 创建struts.xml文件

为了使生成的报表项目支持 Struts 2.0 框架，需要在 ReportJxl/src 目录下创建 struts.xml 文件，该文件的内容如代码 19.6 所示。

代码 19.6　创建 struts.xml 文件：struts.xml

```
...
<struts>
  <!--设定全局参数-->
```

```
    <constant name="struts.i18n.encoding" value="GBK"></constant>
    <constant name="struts.action.extension" value="action,do,itfuture">
    </constant>
    <constant name="struts.locale" value="zh CN"></constant>
    <constant name="struts.devMode" value="true"></constant>
    <constant name="struts.enable.SlashesInActionNames" value="true">
    </constant>

</struts>
```

至此，就完成了对生成报表项目中 Struts 2.0 框架的配置。

19.3.3　关于 Hibernate 3.0 的准备

在实现 Struts 2.x 框架与 Hibernate 3.0 框架二者集成时，对于其中的 Hibernate 3.0 框架除了需要引入相应的 jar 文件外，还必须得对 hibernate.cfg.xml 文件做相应的配置。

1. 引入jar文件

首先引入关于 Hibernate 3.0 框架的核心包：Hibernate 3.0.jar、log4j-1.2.13.jar、cglib-nodep-2.1_3.jar、dom4j-1.6.1.jar、commons-collections.jar、c3p0-0.9.0.4.jar、jta.jar 和 antlr-2.7.6.jar。

2. 创建hibernate.cfg.xml文件

首先通过 MyEclipse 开发环境中关于 Hibernate 框架的向导，让生成报表项目支持 Hibernate 3.0 框架，然后通过该开发环境的反向工程向导实现对数据库 jxl 中两张表进行关系数据库到对象的映射。hibernate.cfg.xml 文件的内容如代码 19.7 所示。

代码 19.7　编辑 hibernate.cfg.xml 文件：hibernate.cfg.xml

```
...
<hibernate-configuration>
    <session-factory>
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <!-- 指定连接数据库的 URL-->
        <property  name="connection.url">jdbc:mysql://localhost:3306/Jxl
        </property>
        <!-- 指定连接数据库用户名-->
        <property name="connection.username">root</property>
        <!-- 指定连接数据库密码-->
        <property name="connection.password">root</property>
        <!-- 指定连接数据库的驱动类-->
        <property name="connection.driver_class">
            com.mysql.jdbc.Driver
        </property>
        <!-- 指定关于数据库的方言->
        <property name="myeclipse.connection.profile">MySQL</property>
        <!-- 指定 Hibernate 映射文件 -->
        <mapping resource="com/cjg/po/Orderitem.hbm.xml" />
        <mapping resource="com/cjg/po/Orderinfo.hbm.xml" />
    </session-factory>
```

```
</hibernate-configuration>
```

至此，就完成了对生成报表项目中 Hibernate 3.0 框架的配置。

19.4　生成报表具体开发——持久层和服务层

为了让读者可以快速理解和掌握生成报表功能，在具体讲解时对该功能按照 MVC 模式分成为领域模型层、持久层、业务层和表现层。由于在前面章节已经介绍了领域模型层，所以本节将介绍关于两个表的持久层和业务层。

19.4.1　关于 orderinfo 表的持久层

在实现关于 orderinfo 表的操作时，因为是由 MyEclipse 开发工具通过反转功能自动生成，所以将不详细介绍。该文件的具体内容如代码 19.8 所示。

代码 19.8　操作 orderinfo 表：OrderinfoDAO.java

```
...
public class OrderinfoDAO extends BaseHibernateDAO {
    private static final Log log = LogFactory.getLog(OrderinfoDAO.class);
    public List findAll(String cond){                          //查找所有记录方法
    log.debug("saving Orderinfo instance");
    String hql = " from Orderinfo orderinfo where 1=1 "+cond;
                                                               //创建 SQL 语句变量
        try {
            return getSession().createQuery(hql).list();//执行 SQL 语句
        } catch (RuntimeException re) {
            log.error("save failed", re);
            throw re;
        }
    }
    public void save(Orderinfo transientInstance) {            //添加记录方法
        log.debug("saving Orderinfo instance");
        try {
            getSession().save(transientInstance);              //添加相应记录
            log.debug("save successful");
        } catch (RuntimeException re) {
            log.error("save failed", re);
            throw re;
        }
    }
    public void delete(Orderinfo persistentInstance) { //删除记录方法
        log.debug("deleting Orderinfo instance");
        try {
            getSession().delete(persistentInstance);      //删除相应记录
            log.debug("delete successful");
        } catch (RuntimeException re) {
            log.error("delete failed", re);
            throw re;
        }
    }
    public Orderinfo findById( java.lang.Integer id) { //查找记录方法
```

```
        log.debug("getting Orderinfo instance with id: " + id);
        try {
            Orderinfo instance = (Orderinfo) getSession()
                    .get("org.wllt.po.Orderinfo", id);    //查找 ID 的相应记录
            return instance;
        } catch (RuntimeException re) {
            log.error("get failed", re);
            throw re;
        }
    }
    public List findByExample(Orderinfo instance) {    //查找记录方法
        log.debug("finding Orderinfo instance by example");
        try {
            List results = getSession()
                    .createCriteria("org.wllt.po.Orderinfo")
                    .add(Example.create(instance))
            .list();                                    //查找相应订单明细的记录
            log.debug("find by example successful, result size: " +
            results.size());
            return results;
        } catch (RuntimeException re) {
            log.error("find by example failed", re);
            throw re;
        }
    }
    public List findByProperty(String propertyName, Object value)
    {          //查找记录方法
      log.debug("finding Orderinfo instance with property: " + propertyName
            + ", value: " + value);
      try {
         String queryString = "from Orderinfo as model where model."
                             + propertyName + "= ?";    //创建 SQL 语句
         Query queryObject = getSession().createQuery(queryString);
                                                        //创建 Query 对象
         queryObject.setParameter(0, value);
          return queryObject.list();
      } catch (RuntimeException re) {
         log.error("find by property name failed", re);
         throw re;
         }
      }
...
```

【代码解析】

上述代码中除了 findAll()方法是由程序员来编写外，其他方法都是由开发环境自动生成的。findAll()方法用来实现查找所有记录的方法。

19.4.2　关于 orderitem 表的持久层

在实现关于 orderitem 表的操作时，因为是由 MyEclipse 开发工具通过反转功能自动生成的，所以将不详细介绍。该文件的具体内容如代码 19.9 所示。

代码 19.9　操作 orderitem 表：OrderitemDAO.java

```
...
public class OrderitemDAO extends BaseHibernateDAO {
```

```
//定义关于类 OrderitemDAO 的变量
private static final Log log = LogFactory.getLog(OrderitemDAO.class);
public void save(Orderitem transientInstance) {
                                            //关于添加 Orderitem()方法
    log.debug("saving Orderitem instance");
    try {
        getSession().save(transientInstance);    //实现保存功能
        log.debug("save successful");
    } catch (RuntimeException re) {
        log.error("save failed", re);
        throw re;
    }
}
public void delete(Orderitem persistentInstance) {
                                        //关于删除 Orderitem()方法
    log.debug("deleting Orderitem instance");
    try {
        getSession().delete(persistentInstance); //实现删除功能
        log.debug("delete successful");
    } catch (RuntimeException re) {
        log.error("delete failed", re);
        throw re;
    }
}
public Orderitem findById( java.lang.Integer id) {
                                        //关于通过 ID 查找方法
    log.debug("getting Orderitem instance with id: " + id);
    try {
        Orderitem instance = (Orderitem) getSession()
                .get("org.wllt.po.Orderitem", id);    //实现查找功能
        return instance;
    } catch (RuntimeException re) {
        log.error("get failed", re);
        throw re;
    }
}
public List findByExample(Orderitem instance) {//关于通过实例查找方法
    log.debug("finding Orderitem instance by example");
    try {
        List results = getSession()
                .createCriteria("org.wllt.po.Orderitem")
                .add(Example.create(instance))    //实现查找功能
        .list();
        log.debug("find by example successful, result size: " +
        results.size());
        return results;
    } catch (RuntimeException re) {
        log.error("find by example failed", re);
        throw re;
    }
}
public List findByProperty(String propertyName, Object value) {
                                            //关于通过属性查找方法
  log.debug("finding Orderitem instance with property: " + propertyName
        + ", value: " + value);
  try {
     String queryString = "from Orderitem as model where model."
                        + propertyName + "= ?";    //设置 SQL 语句
     Query queryObject = getSession().createQuery(queryString);
```

```
                                                            //实现查询功能
        queryObject.setParameter(0, value);
        return queryObject.list();
      } catch (RuntimeException re) {
        log.error("find by property name failed", re);
        throw re;
      }
      }
    public Orderitem merge(Orderitem detachedInstance) {//关于合并属性方法
       log.debug("merging Orderitem instance");
       try {
          Orderitem result = (Orderitem) getSession()
                 .merge(detachedInstance);                //实现合并属性方法
          log.debug("merge successful");
          return result;
       } catch (RuntimeException re) {
          log.error("merge failed", re);
          throw re;
       }
    }
...
```

注意：在生成报表功能中，关于表格 orderitem 的持久层内容都是由 MyEclipse 开发工具通过反转功能自动生成的。

19.4.3　实现生成报表服务层

OrderService.java 文件主要实现了查找所有记录并生成报表方法，该文件的具体内容如代码 19.10 所示。

代码 19.10　生成报表：OrderService.java

```
...
public class OrderService {
    private OrderinfoDAO orderDao;                   //创建属性 orderDao
    public OrderService() {                          //构造方法
        super();
        orderDao = new OrderinfoDAO();
    }
    public OrderinfoDAO getOrderDao() {              //配置属性 orderDao
        return orderDao;
    }
    public void setOrderDao(OrderinfoDAO orderDao) {
        this.orderDao = orderDao;
    }
    public List findAllOrders() throws Exception { //查找所有记录方法
            return orderDao.findAll("");             //调用 findAll()方法
        } catch (Exception e) {
            e.printStackTrace();
            throw e;
        } finally {
            HibernateSessionFactory.closeSession();    //关闭 Session 对象
        }
    }
    //根据 ID 查订单包含订单明细
```

```
public Orderinfo getOrderinfo(Orderinfo order) throws Exception {
    try {
        order = orderDao.findById(order.getOrderid());
                                                //获取相应的 order 对象
        Hibernate.initialize(order.getOrderitems());
        return order;
    } catch (Exception e) {
        e.printStackTrace();
        throw e;
    } finally {
        HibernateSessionFactory.closeSession();    //关闭 Session 对象
    }
}
//为一个 orderinfo 对象创建一个报表
public void createReport(Orderinfo order) throws Exception {
    // 创建存放报表文件夹：report 文件夹
    String filePath = ServletActionContext.getServletContext().
    getRealPath(
            "report");                          //获取 report 文件夹路径
    File path = new File(filePath);             //创建 File 对象
    //判断 report 文件夹是否存在
    if (!path.exists()) {                       //当 report 文件夹不存在
        path.mkdirs();
    }
    //为文件设置文件名
    String fileName = order.getOrdermc() + "-"
            + Dateformat.formatDate(order.getOrderinfodate(), "yyyy-
            MM-dd");
    File eFile = new File(path, fileName + ".xls");
                                                // 创建存放报表文件夹
    if (!eFile.exists()) {
        eFile.createNewFile();
    }
    String[][] orderinfo = {
            { "订单编号", order.getOrderid() + "" },
            { "订单名称", order.getOrdermc() },
            { "订单总数量", order.getTotalnum() + "" },
            { "订单总价", order.getTotalje() + "" },
            {"订单日期",Dateformat.formatDate(order.getOrderin-
            fodate(),"yyyy-MM-dd") },
            { "打印日期", Dateformat.formatDate(new Date(), "yyyy-
            MM-dd") }, };
    //定义 items 的二维数组：
    String[][] orderitems = new String[order.getOrderitems().size() +
    1][];//行为 set 容器的长度 +1 是为表头流出位置
    //第一行是表头信息：
    orderitems[0] = new String[] { "明细编号", "明细名称", "数量", "单价" };
    //循环产生 item 内容组成的 String 一维数组，赋给二维数组的每一行：
    int i = 1;
    for (Orderitem item : (Set<Orderitem>) order.getOrderitems()) {
        orderitems[i++] = new String[] {
                item.getItemid() + "",
                item.getProductmc() ,
                item.getNum() + "" ,
                item.getPrice()+ "" };
    }
    this.writeToFile(eFile, order, orderinfo,orderitems);
}
```

```
//把 orderinfo 写入文件中：
public void writeToFile(File eFile, Orderinfo order, String[][]
orderinfo,String[][] items)
        throws Exception {
    //创建一个 WritableWorkBook，代表这个订单的报表文件 ；
    WritableWorkbook book = Workbook.createWorkbook(eFile);
    //创建一个工作簿
    WritableSheet sheet = book.createSheet(Dateformat.formatDate
    (order.getOrderinfodate(), "MM-dd")+ "订单", 0);
    //设置各种文字格式
    WritableFont titleFont = new WritableFont(WritableFont.ARIAL, 20,
            WritableFont.BOLD, false);             //设置标题格式
    WritableFont sTitleFont = new WritableFont(WritableFont.ARIAL, 12,
            WritableFont.BOLD, false);             //设置字段格式
    WritableFont contFont = new WritableFont(WritableFont.ARIAL, 12,
            WritableFont.NO_BOLD, false);          //设置内容格式
    //生成各种单元格格式：
    WritableCellFormat titleFormat = new WritableCellFormat
    (titleFont);
    WritableCellFormat sTitleLeft = new WritableCellFormat
    (sTitleFont);
    WritableCellFormat sTitleCentre = new WritableCellFormat
    (sTitleFont);
    WritableCellFormat conFormatLeft = new WritableCellFormat
    (contFont);
    WritableCellFormat conFormatCentre = new WritableCellFormat
    (contFont);
    //调整单元格格式：
    titleFormat.setAlignment(Alignment.CENTRE);
    titleFormat.setVerticalAlignment(VerticalAlignment.CENTRE);
    titleFormat.setBorder(Border.NONE, BorderLineStyle.NONE);
    sTitleLeft.setAlignment(Alignment.LEFT);
    sTitleLeft.setVerticalAlignment(VerticalAlignment.CENTRE);
    sTitleLeft.setBorder(Border.NONE, BorderLineStyle.NONE);
    sTitleCentre.setAlignment(Alignment.CENTRE);
    sTitleCentre.setVerticalAlignment(VerticalAlignment.CENTRE);
    sTitleCentre.setBorder(Border.ALL, BorderLineStyle.THIN);
    conFormatLeft.setAlignment(Alignment.LEFT);
    conFormatLeft.setVerticalAlignment(VerticalAlignment.CENTRE);
    conFormatLeft.setBorder(Border.NONE, BorderLineStyle.THIN);
    conFormatCentre.setAlignment(Alignment.CENTRE);
    conFormatCentre.setVerticalAlignment(VerticalAlignment.CENTRE);
    conFormatCentre.setBorder(Border.ALL, BorderLineStyle.THIN);
    //合并单元格
    sheet.mergeCells(0, 0, items[0].length-1, 0);
    //调整列宽：
    sheet.setColumnView(0, 14);
    sheet.setColumnView(1, 14);
    sheet.setColumnView(2, 14);
    sheet.setColumnView(3, 14);
    Label titleLabel = new Label(0, 0, Dateformat.formatDate(order
            .getOrderinfodate(), "MM 月 dd 日")
            + "订单报表", titleFormat);                 //创建报表标题
    sheet.addCell(titleLabel);

    for (int i = 0; i < orderinfo.length; i++) {       //写入订单信息
```

```
            for (int j = 0; j < orderinfo[i].length; j++) {
                Label label = new Label(j, i + 2, orderinfo[i][j],
                        j == 0 ? sTitleLeft : conFormatLeft);
                sheet.addCell(label);
            }
        }
        for(int i=0;i<items.length;i++){                    //写入订单明细信息
            for(int j=0;j<items[i].length;j++){
                Label label = new Label(j,i+9,items[i][j],i==0?
                sTitleCentre:conFormatCentre);
                sheet.addCell(label);
            }
        }
        book.write();                                      //真正的写入内容
        book.close();                                      //关闭资源
    }
}
...
```

【代码解析】

在关于 JXL 组件的 API 中，类 WritableWorkbook 对应于 Excel 文件，该类的具体定义如下：

```
Public abstract class WritableWorkbook()
```

在该类中存在 3 个经常使用的方法：获得工作簿（Workbook）中工作表（Sheet）的个数——getNumberOfSheets()；返回工作簿（Workbook）中工作表（Sheet）对象数组——getSheets()；用指定的名称和标号产生一个工作簿——createSheet()。

注意：如果想获取 WritableWorkbook 对象，可以通过工厂类 Workbook 来实现，该类的定义如下：

```
Public abstract class Workbook()
```

在关于 JXL 组件的 API 中存在许多设置格式的方法，它们分别为：

```
WritableFont(WritableFont.FontName    fn,int    ps,WritableFont.BoldStyle
bs,boolean italic)
```

上述代码用指定的字体名称、字号生成字体，并指定粗体格式，是否斜体。

```
Public class Alignment()
```

上述类用来实现水平对齐方式，其值可以是 CENTRE/LEFT/RIGHT/JUSTIFY 等。

```
Public class VerticalAlignment()
```

上述类用来实现垂直对齐方式，其值可以是 CENTRE/TOP/BOTTOM/JUSTIFY 等。

```
Public class Border()
```

上述类用来设置单元格边框，其值可以是 ALL/TOP/BOTTOM/LEFT/RIGHT/NONE 等

```
Public class BorderLineSytly()
```

上述类用来设置边框线，其值可以是 THIN/THICK/NONE/DOUBLE 等。

19.5　生成报表具体开发——表示层

为了让读者可以快速地理解和掌握生成报表功能，在具体讲解时对该功能按照 MVC 模式分成为领域模型层、持久层、业务层和表现层。由于在前面章节已经介绍了业务层，所以本节将介绍关于生成报表功能的表示层。

19.5.1　实现页面跳转 Action 类

生成报表系统中，所有请求都由 Struts 2.0 框架中名为 OrderAction 的 Action 类来处理，该类的具体内容如代码 19.11 所示。

代码 19.11　页面跳转：OrderAction.java

```
...
public class OrderAction {
    private List orderinfos;                                //字段 orderinfos
    private Orderinfo orderinfo;                            //字段 orderinfo
    private OrderService service;                           //字段 service
    //省略属性 orderinfos、orderinfo 和 service 的配置
    ...
    public OrderAction() {                                  //构造函数
        super();
        service = new OrderService();
    }
    public String findAllOrder() throws Exception { //获得所有的订单信息
        orderinfos = service.findAllOrders();   //调用 findAllOrders()方法
        return "success";
    }
    //根据 ID 得到一个订单及这个订单中的订单明细
    public String findAOrder() throws Exception {
        orderinfo = service.getOrderinfo(orderinfo);
                                               //调用 getOrderinfo()方法
        return "success";
    }
    //根据 ID 得到一个订单及这个订单中的订单明细，并生成 excel 报表;
    public String createOrderReport() throws Exception {
        orderinfo = service.getOrderinfo(orderinfo);
                                               //调用 getOrderinfo()方法
        service.createReport(orderinfo);       //创建报表
        return "success";
    }
}
```

为了方便管理，将关于 Struts 2.0 框架的配置信息存放在 struts.xml 配置文件里，具体内容如代码 19.12 所示。而在 actions.xml 文件中加入该 Action 类的配置，具体内容如代码 19.13 所示。

代码 19.12　关于 Struts 2.0 框架配置文件：actions.xml

```
...
<struts>
```

```
    <constant name="struts.i18n.encoding" value="GBK"></constant>
    <constant name="struts.action.extension" value="action,do,itfuture">
    </constant>
    <constant name="struts.locale" value="zh CN"></constant>
    <constant name="struts.devMode" value="true"></constant>
    <constant  name="struts.enable.SlashesInActionNames"  value="true">
    </constant>
    <!--引入相应的 action 文件 -->
    <include file="actions.xml"></include>
</struts>
```

代码 19.13　关于 Action 类配置文件：actions.xml

```
...
<struts>
  <package name="show" namespace="/show" extends="struts-default">
    <!--发生异常时，转向的页面-->
    <global-results>
        <result name="exception">/exception.jsp</result>
    </global-results>
    <!--设置异常-->
    <global-exception-mappings>
        <exception-mapping result="exception" exception="java.lang.
        Exception"></exception-mapping>
    </global-exception-mappings>
    <!-- 配置名为 showOrders 的 Action-->
    <action name="showOrders" class="com.cjg.action.OrderAction" method=
    "findAllOrder">
      <result name="success">/showOrder.jsp</result>
    </action>
    <!-- 配置名为 createReport 的 Action-->
    <action name="createReport" class="com.cjg.action.OrderAction"
    method= "createOrderReport">
      <result name="success">/showOrder.jsp</result>
    </action>
  </package>
</struts>
```

注意：在上述代码中，当关于 showOrders 的请求处理成功后，就会转到显示订单列表的页面 showOrder.jsp。

19.5.2　关于生成报表的页面

通过 actions.xml 文件的相关配置可以发现，生成报表系统需要两个页面：showOrder.jsp 和 exception.jsp。除了这两个页面外，还需要一个欢迎页面 index.jsp。

1．欢迎页面

index.jsp 页面作为生成报表系统的欢迎页面，用来发出关于展示订单的请求 showOrders.action，该页面的具体内容如代码 19.14 所示。

代码 19.14　首页：index.jsp

```
...
  <body>
```

```
    <a href="show/showOrders.action">展现所有订单</a>   <!--链接-->
  </body>
...
```

【代码解析】

在上述代码中，href 的值之所以是 show/showOrders.action，而不是/showOrders.action，是因为在 actions.xml 文件中，showOrders.action 请求是配置在<package name="show">中。

2. 显示订单信息页面

showOrder.jsp 页面用来显示订单信息记录，该页面的具体内容如代码 19.15 所示。

代码 19.15　显示订单信息：showOrder.jsp

```
...
  <body>
    <table align="center" border="1">
    <tr align="center">                       <!--表头标题-->
        <td>订单编号</td>
        <td>订单名称</td>
        <td>订单总数量</td>
        <td>订单金额</td>
        <td>日期</td>
        <td>操作</td>
    </tr>
      <s:iterator value="orderinfos">        <!--遍历对象，显示相关信息-->
      <tr align="center">
        <td><s:property value="orderid"/></td>
        <td><s:property value="ordermc"/></td>
        <td><s:property value="totalnum"/></td>
        <td><s:property value="totalje"/></td>
        <td><s:property value="orderinfodate"/></td>
        <td><a href="show/createReport.action?orderinfo.orderid=$
        {orderid}">生成报表</a></td>
    </tr>
      </s:iterator>
    </table>
  </body>
...
```

【代码解析】

在上述代码中，当单击“生成报表”链接就会发出关于生成报表的请求——createReport.action。

3. 异常页面

exception.jsp 页面是生成报表的出错页面，该页面的具体内容如代码 19.16 所示。

代码 19.16　异常页面：exception.jsp

```
...
  <body>
    这是一个异常页面．<br>
    <s:property value="exception"/>                <!--显示出错信息-->
```

```
  </body>
...
```

19.5.3　实现生成报表工具类

在生成报表系统中需要两个工具类：HibernateSessionFactory.java 和 Dateformat.java，前者用来创建 Session 对象，具体内容如代码 19.17 所示。后者主要用来设置时间的格式，具体内容如代码 19.18 所示。

代码 19.17　创建 Session 工厂：HibernateSessionFactory.java

```
...
public class HibernateSessionFactory {
    //创建各种变量
    private static String CONFIG FILE LOCATION = "/hibernate.cfg.xml";
    private   static   final   ThreadLocal<Session>   threadLocal   =   new
    ThreadLocal<Session>();
    private static Configuration configuration = new Configuration();
    private static org.hibernate.SessionFactory sessionFactory;
    private static String configFile = CONFIG FILE LOCATION;
    private HibernateSessionFactory() {                //工厂类
    }
    public static Session getSession() throws HibernateException {
                                                       //获取 Session 对象
        Session session = (Session) threadLocal.get(); //获取 Session 对象
        if (session == null || !session.isOpen()) {    //判断 Session 对象
            if (sessionFactory == null) {
                rebuildSessionFactory();
            }
            //为变量 session 赋值
            session = (sessionFactory != null) ? sessionFactory.
            openSession()
                    : null;
            threadLocal.set(session);
        }
        return session;
    }
    public static void rebuildSessionFactory() {//创建 SessionFactory 对象
        try {
            configuration.configure(configFile);   //读取配置文件
            sessionFactory = configuration.buildSessionFactory();
                                                   //获取 SessionFactory
        } catch (Exception e) {
            System.err.println("%%%% Error Creating SessionFactory %%%%");
            e.printStackTrace();
        }
    }
    public static void closeSession() throws HibernateException {
                                                   //关闭 Session 对象
        Session session = (Session) threadLocal.get();
        threadLocal.set(null);
        if (session != null) {
            session.close();
        }
```

```
    }
    public static org.hibernate.SessionFactory getSessionFactory() {
                                                          //设置 tSessionFactory 属性
        return sessionFactory;
    }
    public static void setConfigFile(String configFile) {
                                                          //设置配置文件顺序
        HibernateSessionFactory.configFile = configFile;
        sessionFactory = null;
    }
    public static Configuration getConfiguration() {
        return configuration;
    }
}
```

【代码解析】

上述代码是在 MyEclipse 开发环境中通过添加支持 Hibernate 3.0 框架时自动生成的，之所以要生成该文件，是因为在其他层的代码中使都用了 Session 对象。

代码 19.18　关于时间的格式：Dateformat.java

```
...
public class Dateformat {
    public static String formatDate(Date date,String format){
        SimpleDateFormat sdf = new SimpleDateFormat(format);//设置时间格式
        return sdf.format(date);
    }
}
```

19.6　多学两招——其他报表插件

在具体开发生成报表项目的系统时，除了可以使用 JXL 组件，还可以使用 Jakarta POI 组件和 JasperReport 组件。Jakarta POI 组件是 Apache 组织的一款产品，而 JasperReport 是 JasperSoft 公司的一款开源的报表解决方案。

19.6.1　报表插件——Jakarta POI

久负盛名的报表插件——Jakarta POI，提供读写微软 Office 文档格式的方法。该控件不仅支持对 MS Excel 公式的支持、MS Word 文档图像抽取，而且还支持 PowerPoint 文档。本节将从下载 Jakarta POI 组件开始，具体步骤如下。

（1）首先访问 Apache 组织的官方网站（http://www.apache.org/），如图 19.13 所示。在该页面中单击 Foundation 导航栏就可以进入 Apache 组织的产品列表。

（2）打开进入 Apache 组织的产品列表页面（如图 19.14 所示）后，单击右边 Apache Projects 栏目中的 POI 链接就可以进入关于该组件的页面。

（3）在关于 POI 组件的页面中（如图 19.15 所示），单击 HTTP 目录中 http://apache.freelamp.com/poi 链接就可以进入关于下载该组件的页面。

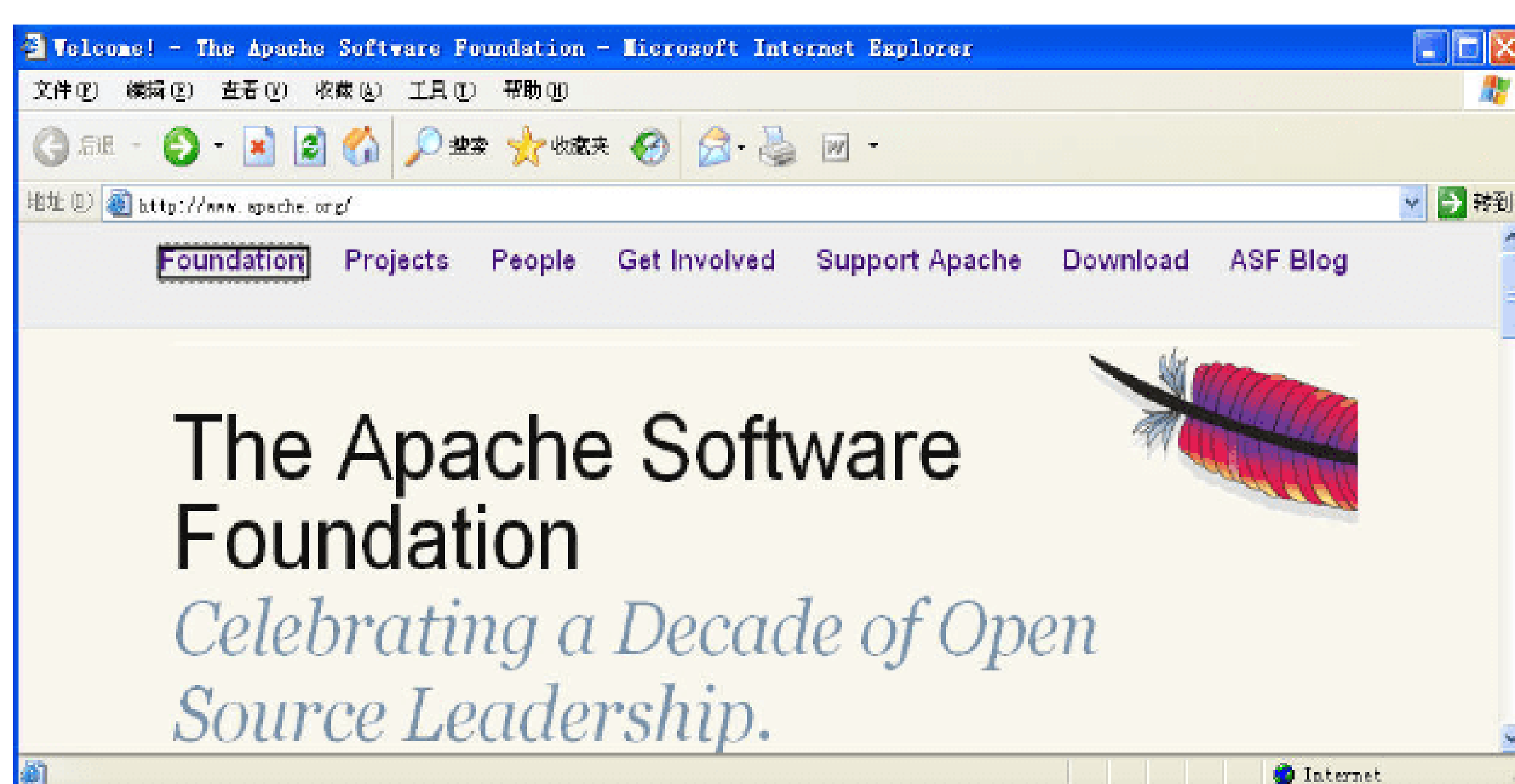


图 19.13　官方首页

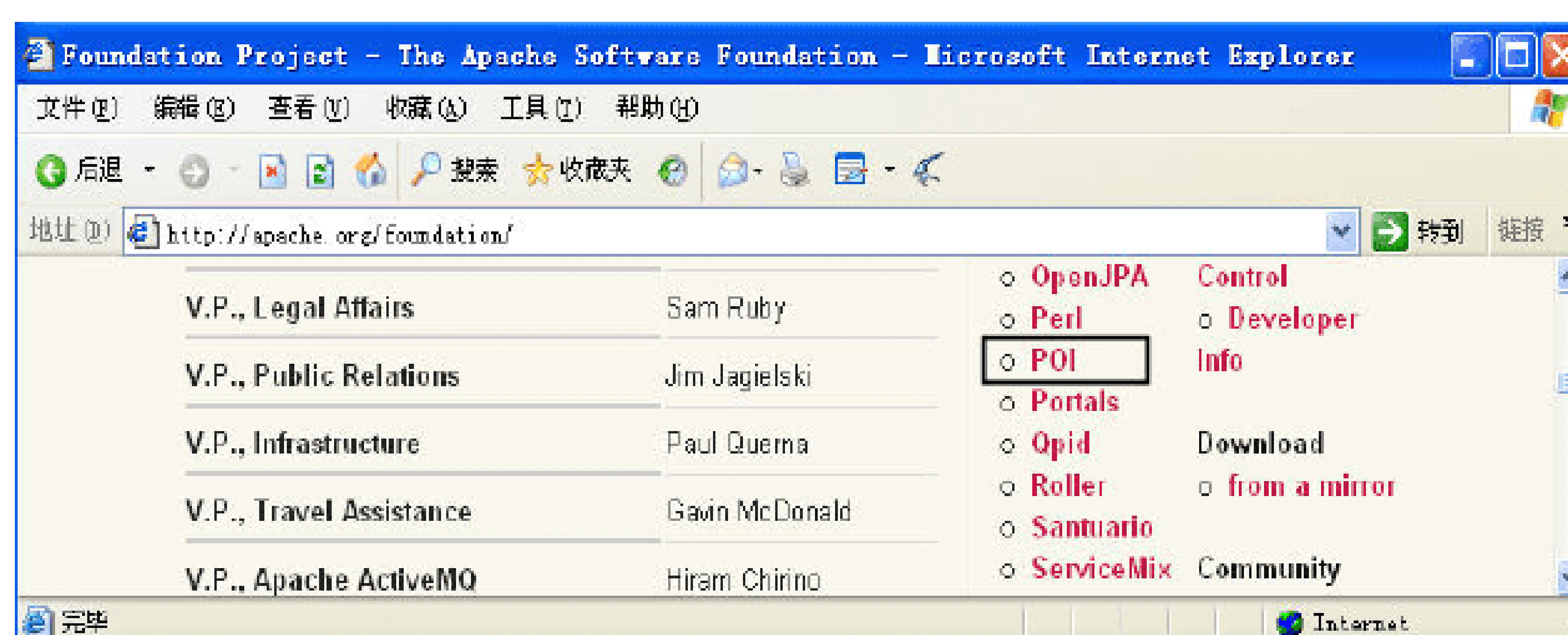


图 19.14　Apache 组织的产品列表

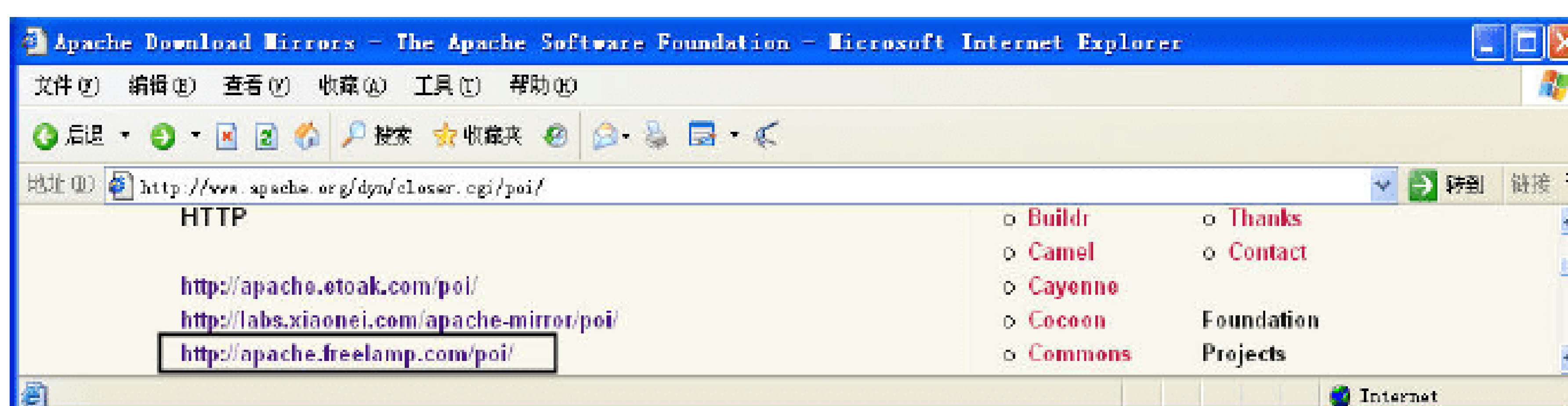


图 19.15　关于 POI 组件页面

至此，就完成了对 Jakarta POI 组件的下载。

19.6.2　报表插件——JasperReport

JasperReport 插件是一个灵活、功能强大的报表制作工具，该插件不仅支持 PDF、HTML 形式生成报表，而且还支持 XML 形式。同时 JasperReport 插件支持的数据源也是多样的，不仅可以是关系数据库而且还可以是 Java 容器对象。本节将从下载 JasperReport 组件开始，具体步骤如下。

（1）首先访问下载 JasperReport 组件的官方网站（http://jasperforg.org/），如图 19.16 所示。在该页面中，分别选择 JasperReports 和 iReport 两个图标来下载相应的软件。

（2）当单击 JasperReports 图标后，就会转到关于 JasperReports 组件类型的页面（如图 19.17 所示）。在该页面中单击 DOWNLOAD 按钮就会转到关于下载 JasperReports 组件的页面。

（3）在关于下载 JasperReports 的页面（如图 19.18 所示）中单击 jasperreports-3.5.0-project.zip 链接，就可以下载该类型的 JasperReports 组件。

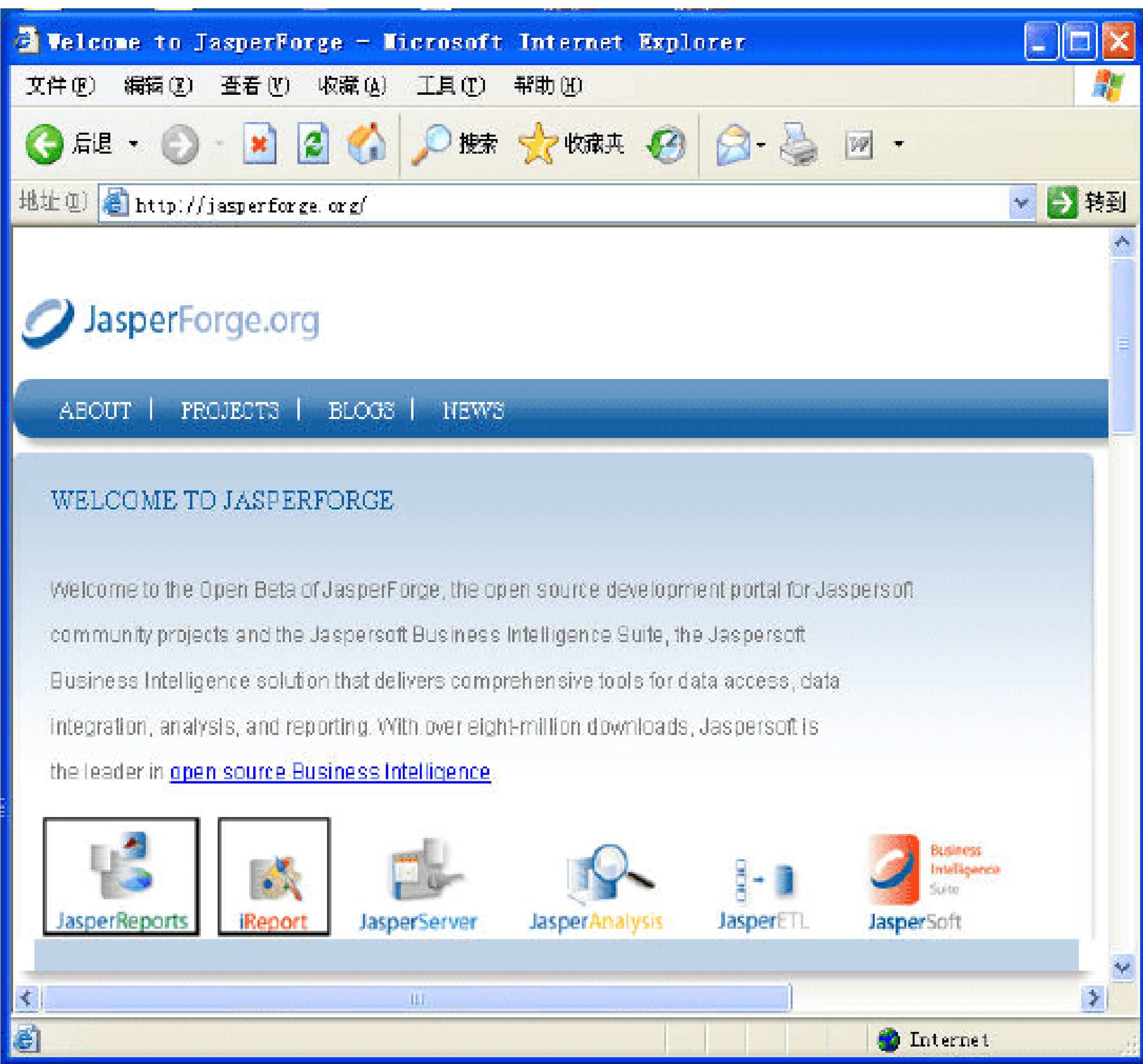


图 19.16　首页

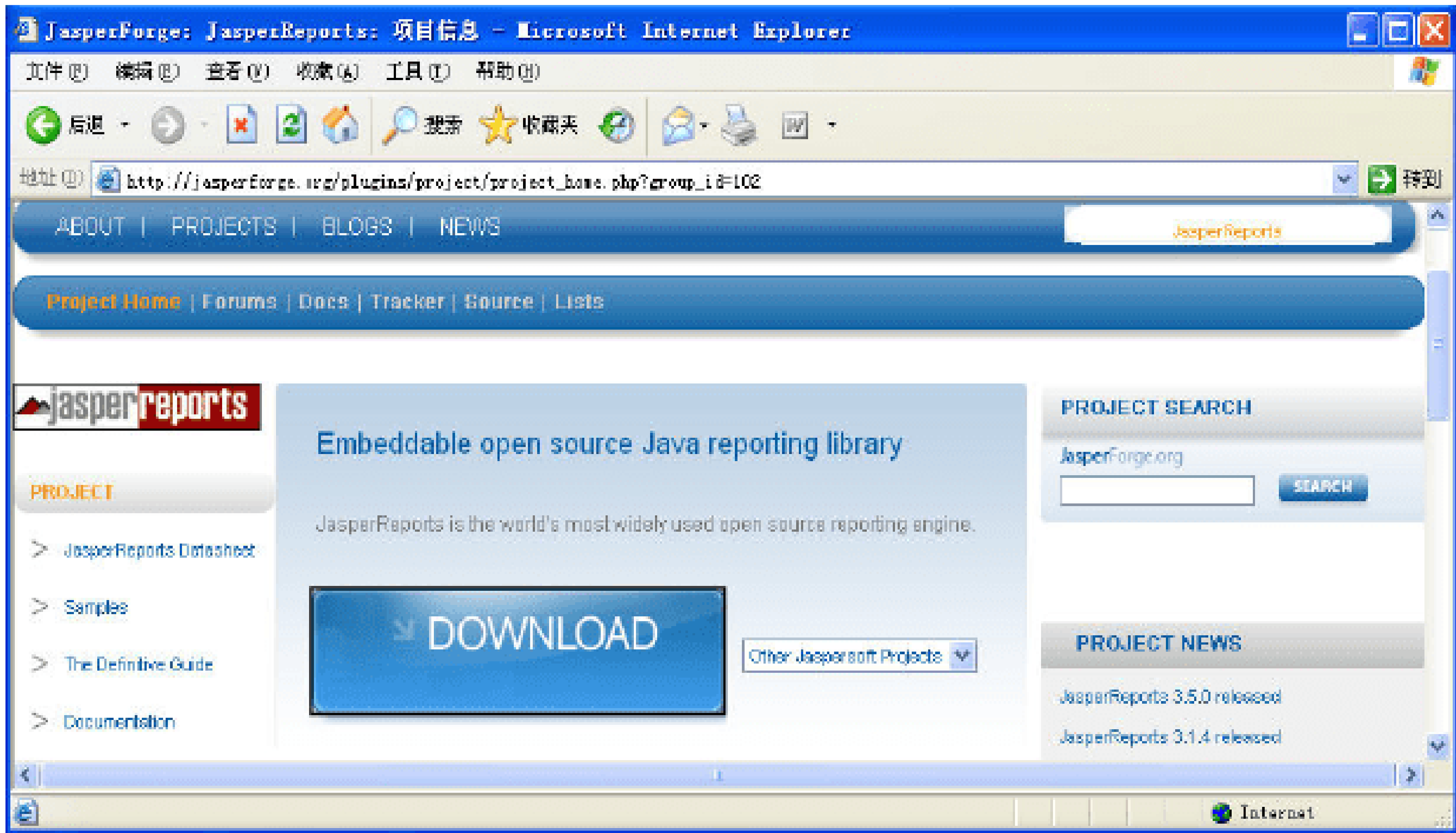


图 19.17　关于 JasperReports 页面

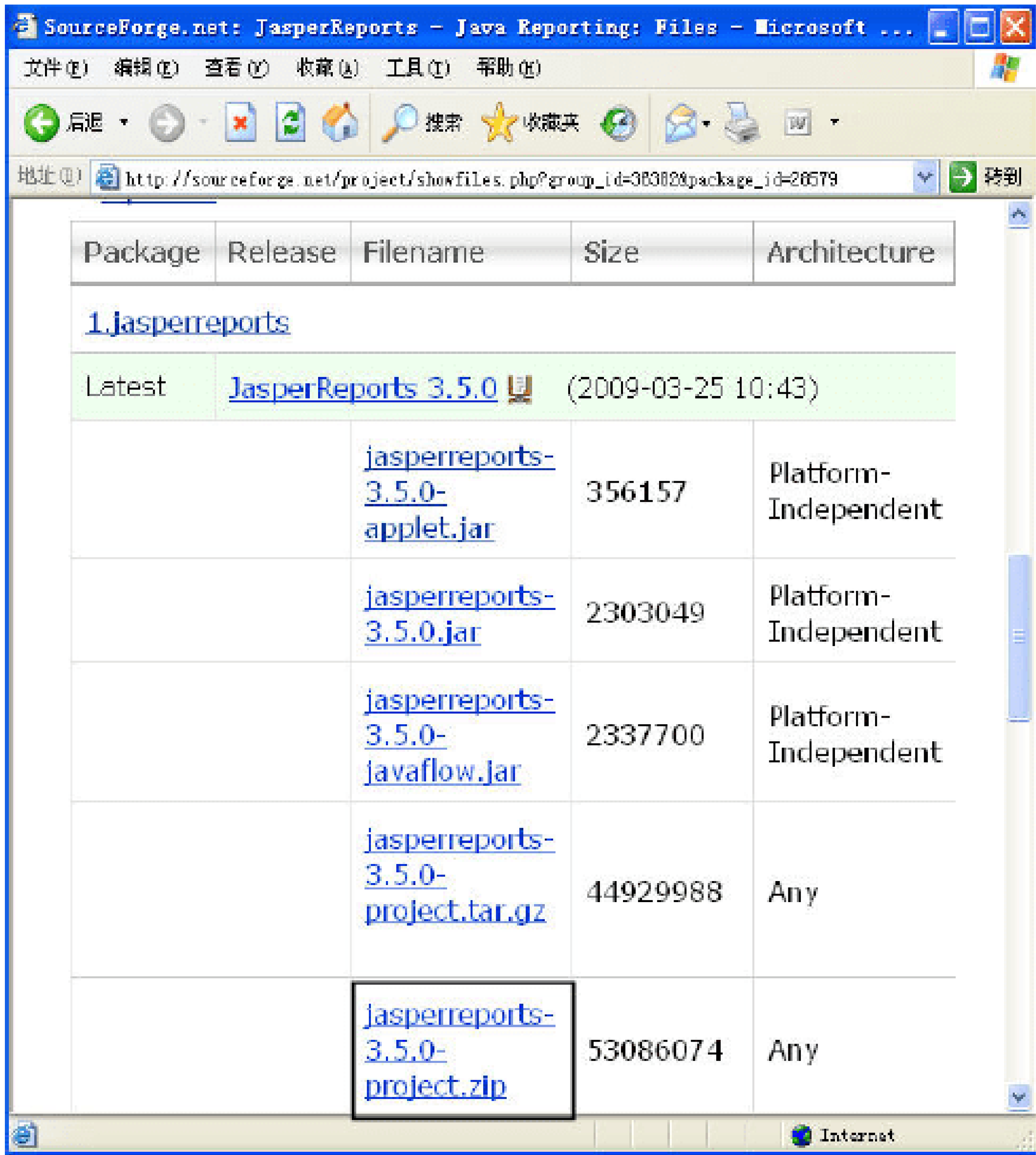


图 19.18　JasperReports 组件

（4）当单击 iReport 图标后，就会转到关于 iReport 软件的页面，如图 19.19 所示。在

该页面中单击 DOWNLOAD 图标就转到关于下载各种版本 iReport 软件的页面。

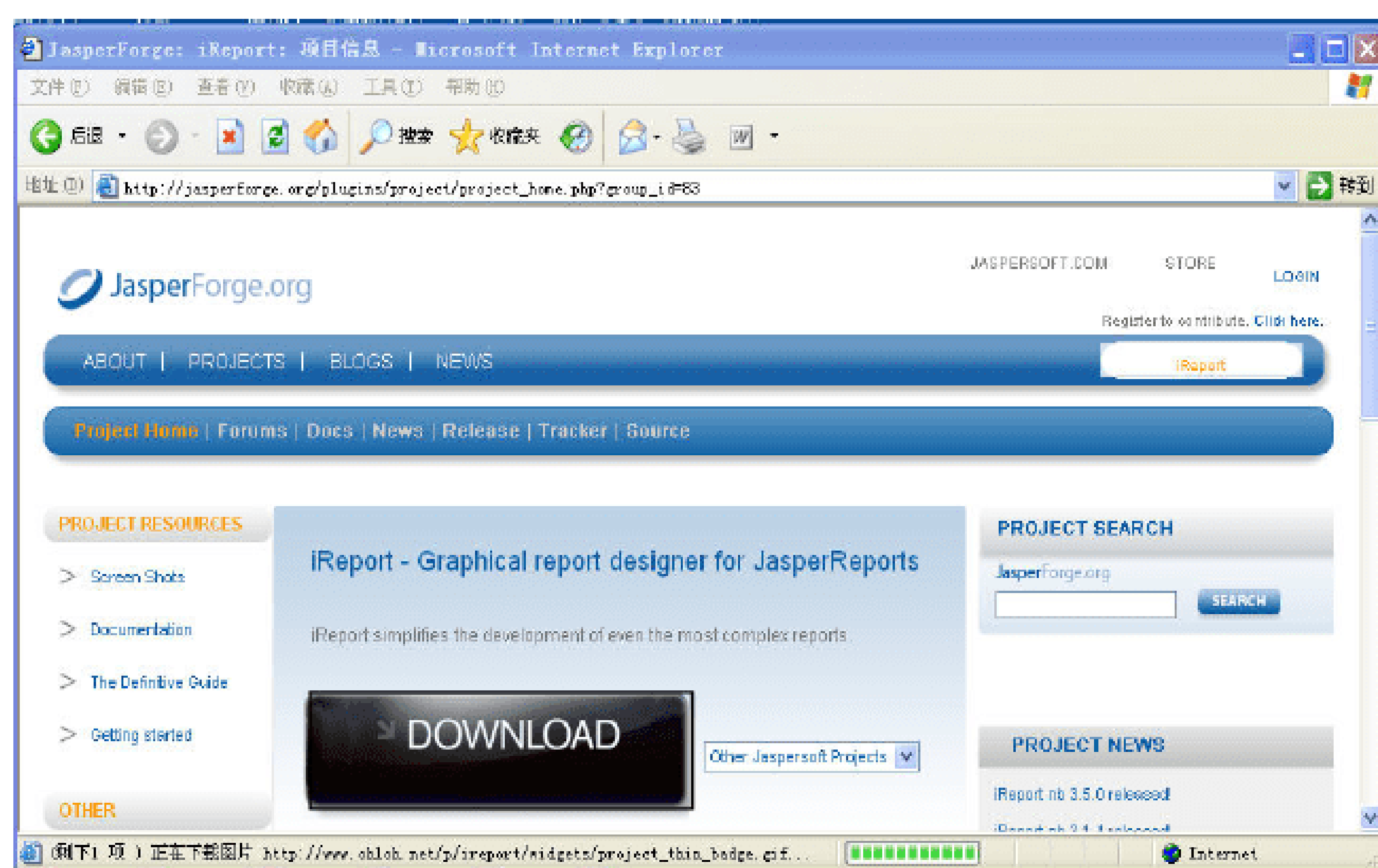


图 19.19　关于 iReport 页面

（5）在各种版本 iReport 软件的页面中（如图 19.20 所示），单击 iReport-nb-3.5.0 记录中 Download 链接就可以转到关于相应软件版本的页面，如图 19.21 所示。在关于 iReport3.5.0 版本软件页面中，单击 iReport-nb-3.5.0.zip 链接就可以下载 iReport 软件。

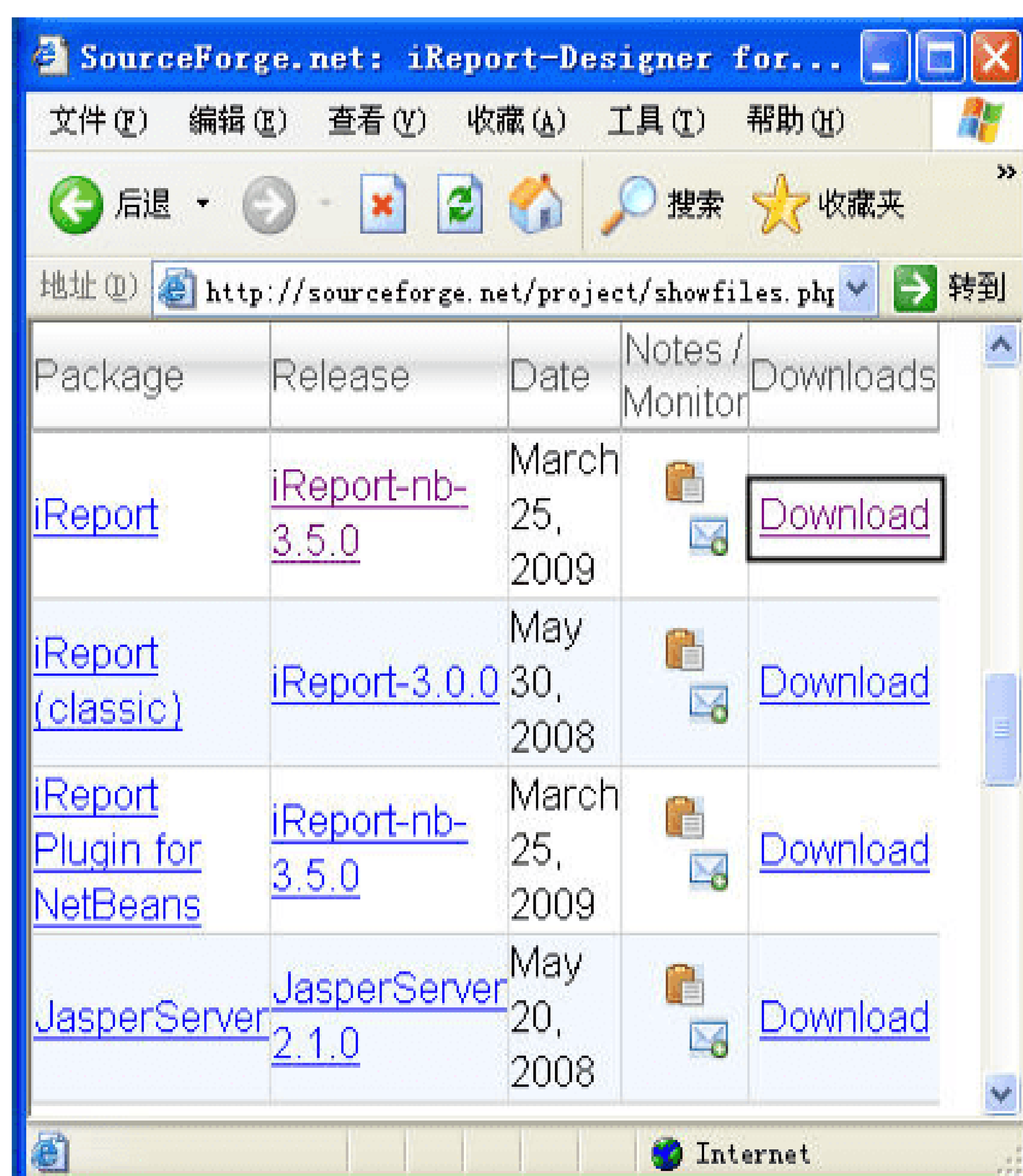


图 19.20　选择 3.5.0 版本

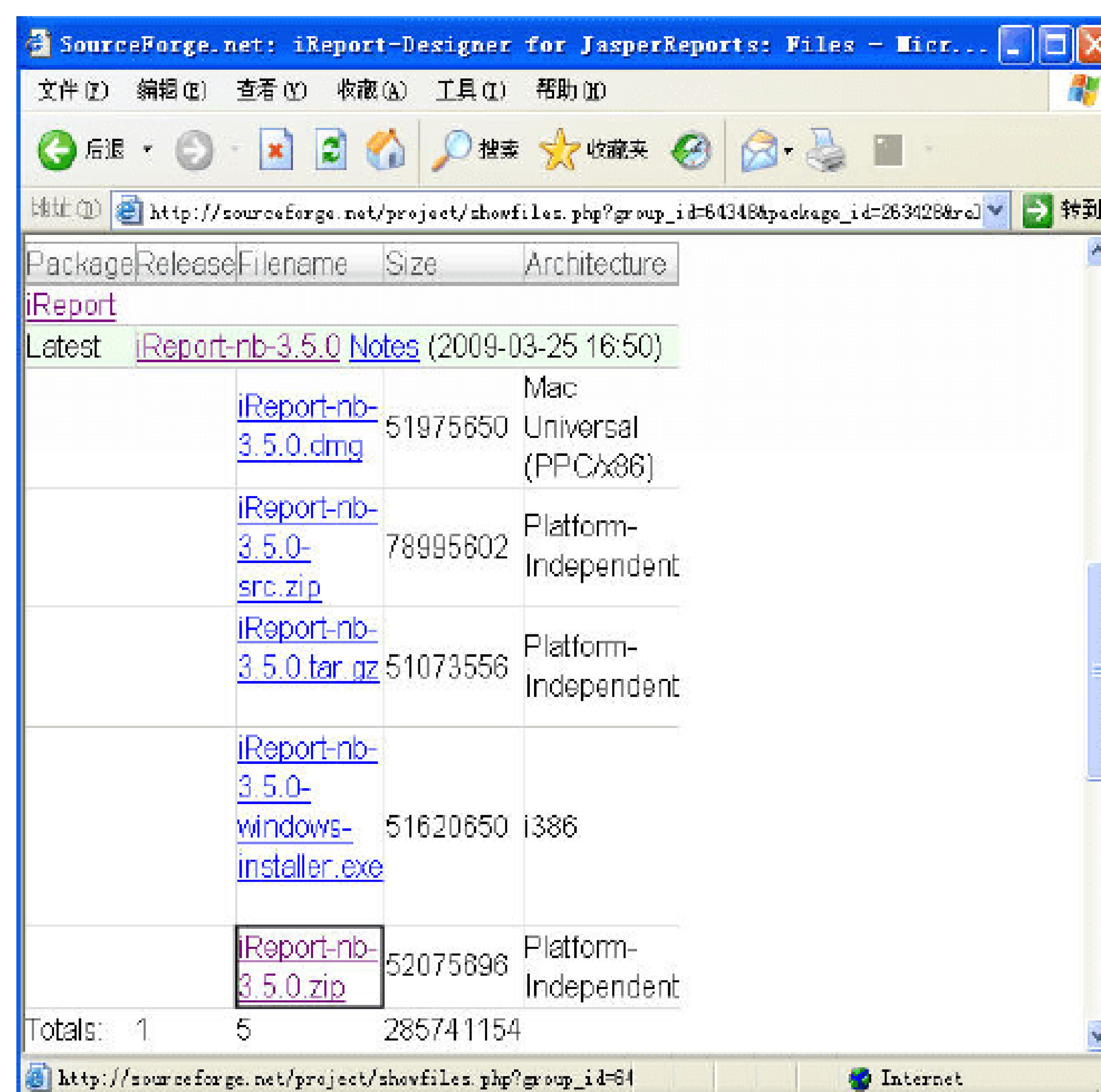


图 19.21　关于 3.5.0 版本的页面

至此，就完成了对 JasperReports 组件的下载。

19.7　小　　结

本章主要介绍生成报表功能，该模块基于 Struts 2.0+Hibernate 3.0+JXL 解决方案，保持了良好的 MVC 分层思想。生成报表项目在业务上主要为显示数据库数据记录和生成关于数据库记录的报表，在具体实现显示数据库数据记录时，主要通过 Hibernate 3.0 框架来实现该功能。而在具体实现关于数据库记录的报表时，却通过 Struts 2.0+JXL 框架来实现该功能。本章除了详细讲解生成报表项目外，还讲解了报表组件的详细使用。

第 20 章　数据格式转换（Struts 2.x+Hibernate+Dom4j）

一个功能强大的网站系统经常会遇到这种情形，需要把数据存储到 XML 文件里。之所以需要该功能，因为当各个系统间数据需要相互转换时，数据必须要存储到 XML 文件中，甚至通过 XSL 语言可以更加友好地显示 XML 文件中存储的数据。

本章将详细地介绍如何利用 Struts 2.0+Hibernate 3.0+Dom4j 框架实现数据库与 XML 文件中数据的相互转换。为了能够更好地理解该功能，本章还将详细介绍如何利用 Dom4j 和 Sax 组件来操作 XML 文件。

20.1　关于 XML 文件基础知识

在 Java Web 项目中可以通过各种方式来操作 XML 文件，例如除了比较常见的 Dom4j 组件和 Sax 组件外，还有 JDom 组件和 Dom 组件。本节除了介绍关于 XML 文件的基础知识外，还将详细地演示数据存储方式的转换功能。

20.1.1　为什么要使用 XML 文件

XML 语言是互联网组织（W3C）为了方便软件开发人员和内容创建者在网页上组织信息而创建的一组规范。利用该语言，可以确保网络进行信息交互合作时，具有良好的可靠性和交互操作性。

在关于 Java Web 项目的各种框架中，经常会遇到作为配置文件的 XXX.xml 文件。例如 Struts 2.0 框架的 struts.xml 配置文件、Spring 框架的 applicationContext.xml 配置文件和 Hibernate 框架的 hibernate.cfg.xml 配置文件。之所以使用 XXX.xml 而不是 XXX.properties 作为配置文件，主要因为：

- XML 文件可以通过 DTD 文件定义其格式，例如 XML 文件包含哪些元素、元素拥有哪些子元素、元素拥有哪些属性及属性的值；
- XML 文件在结构上具有层次感，便于阅读和修改；
- XML 文件具有统一的标准，使得任何编程语言和任何框架都可以解析该类型文件。

当 XML 文件被用来存储数据时，具有其他存储数据手段：数据库、文本文件等不具有的特点，例如：

- XML 文件中的数据可以被不同系统进行传递；
- XML 文件中的数据可以利用 XSL 语言进行友好的显示。

当在开发中小型分布系统时，就需要使用 Apache 组织的 Axis 产品或 Codehase 组织的 Xfire 产品。这两个产品都符合 Web Service 技术规范，而该规范技术核心描述语言 WSDL 就是由 XML 语言组成的。

综上所述，XML 文件在存储数据、携带数据和交互数据方面是其他任何类型文件不可替代的。

20.1.2　数据格式转换功能的描述

本节将以直观的方式来向读者介绍整个数据格式转换模块要实现的功能。为了方便讲解，将通过表部门（dept）与 XML 文件中的数据相互转换的具体实例来介绍数据格式转换模块的各个功能。这些功能包括上传 XML 文件实现导入数据库和数据库记录转换成 XML 文件功能。

1．上传XML文件实现导入数据库功能

首先通过访问地址 http://localhost:8081/XMLData/loadxml.jsp 打开上传文件对话框，如图 20.1 所示。在该对话框中通过单击“浏览”按钮选择 XML 文件的路径后，然后通过单击“提交”按钮就可以实现该 XML 文件的上传，如图 20.2 所示。在 XML 文件成功上传后，除了会存储到根目录 XML 文件中以外（如图 20.3 所示），还会自动把 XML 文件内容存储到数据库中。

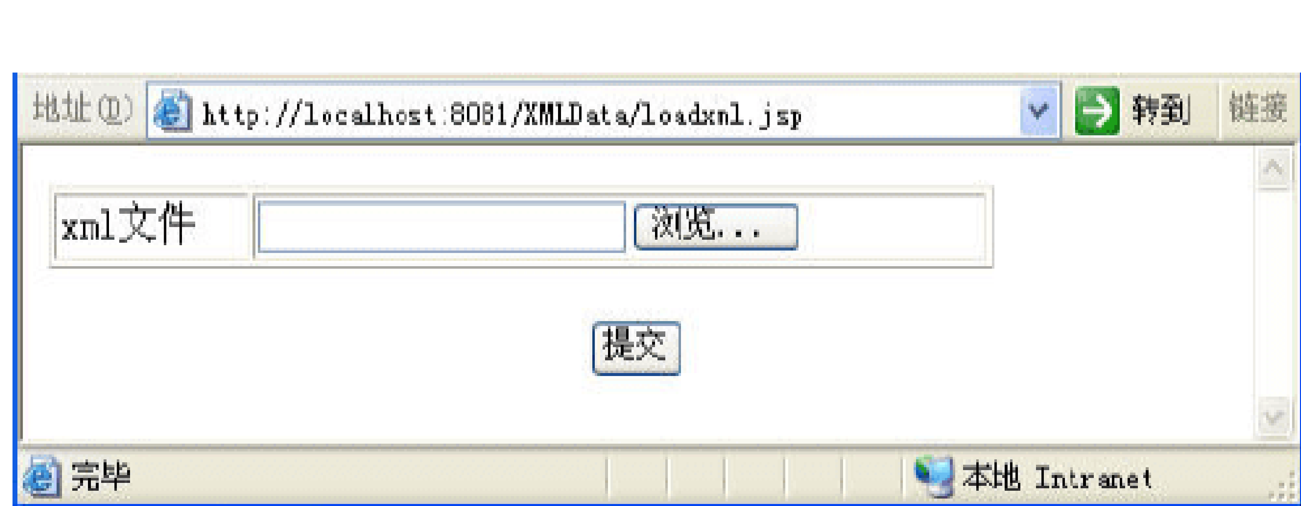


图 20.1　上传文件页面



图 20.2　实现文件的上传

当 XML 文件上传成功后，就会直接转到如图 20.4 所示的显示数据库所有记录的页面。在 XML 文件没上传之前，数据库中表 dept 的数据记录如图 20.5 所示。但是在上传完 XML 文件后，表格 dept 的数据记录却如图 20.6 所示多了一条上传 XML 文件的内容。

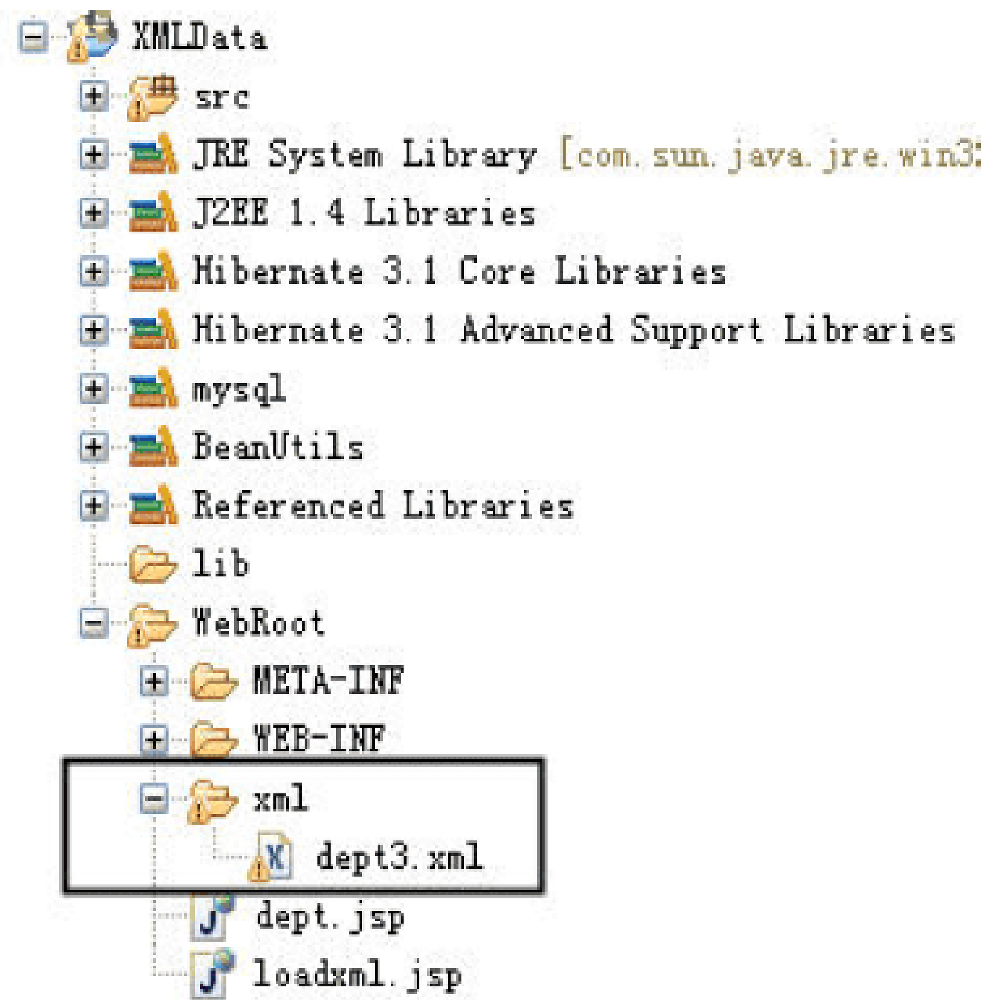


图 20.3　存储位置



部门编号	部门名称	部门编制	部门描述	操作
1	人事部门	1	管理人员	产生xml
2	资源部门	2	分配资料	产生xml
3	录入部门	4	演讲部门	产生xml

图 20.4　显示数据库记录

	DEPTID	DEPTNAME	DEPTNUM	DEPTDESC
□	1	人事部门	1	管理人员
□	2	资源部门	2	分配资料
*				(NULL)

图 20.5　原始表格内容

	DEPTID	DEPTNAME	DEPTNUM	DEPTDESC
□	1	人事部门	1	管理人员
□	2	资源部门	2	分配资料
□	3	录入部门	4	演讲部门
*				(NULL)

图 20.6　上传完后表格内容

2. 数据库记录转换成XML文件

如果想实现数据库记录转换成 XML 文件，可以单击图 20.7 中“操作”字段中的“产生 xml”链接。单击第一条记录的“产生 xml”链接，则可以根据该数据库记录生成 XML 文件，如图 20.7 所示。存储 XML 文件目录如图 20.8 所示。



部门编号	部门名称	部门编制	部门描述	操作
1	人事部门	1	管理人员	产生xml
2	资源部门	2	分配资料	产生xml
3	录入部门	4	演讲部门	产生xml

xmlTODB

图 20.7　生成 XML 文件

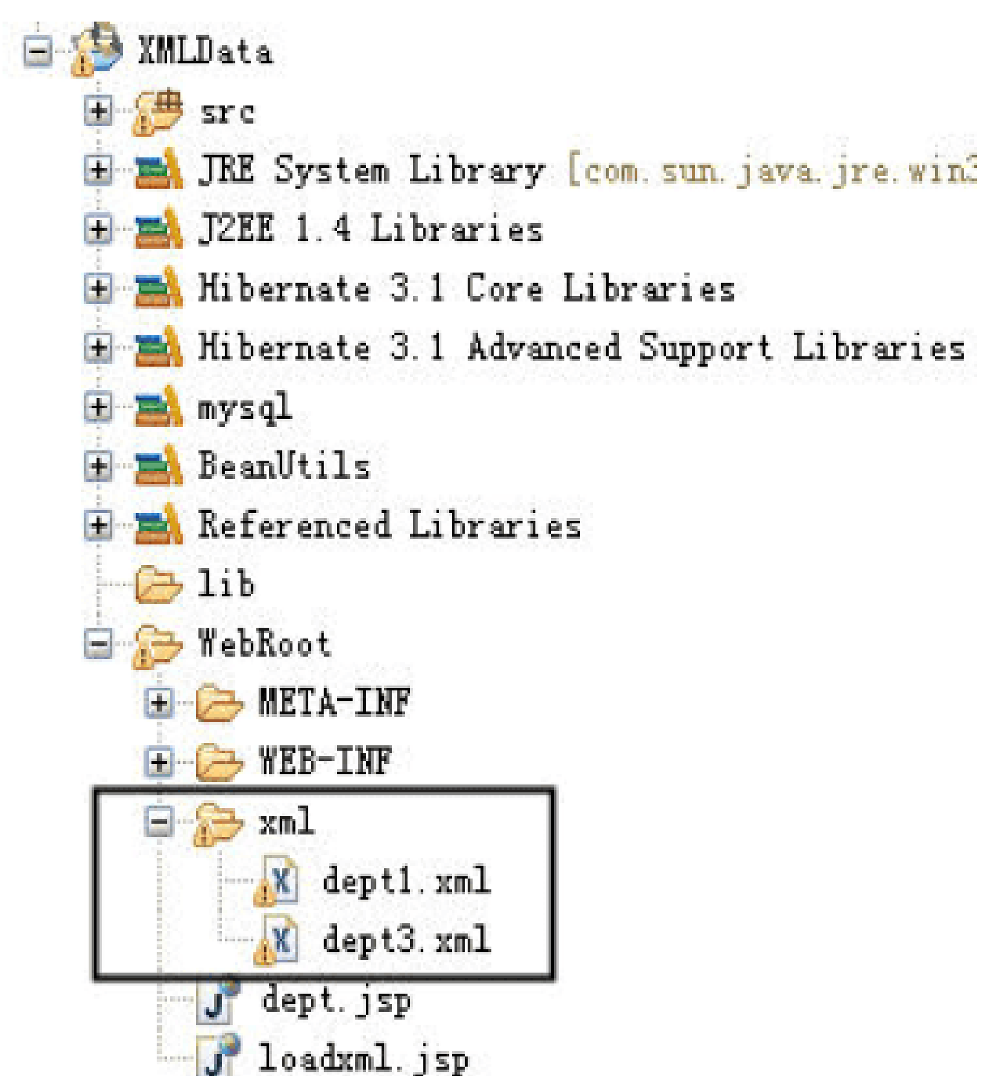


图 20.8　存储 XML 文件目录

20.2　下载 Dom4J

到目前为止，如果想利用基于树的思想来操作 XML 文件，可以使用 Dom、JDom 和 Dom4J 组件。由于大名鼎鼎的 Hibernate 框架使用 Dom4J 组件来读取名为 hibernate.cfg.xml 的 XML 配置文件，所以本章的数据存储方式转换功能也是使用 Dom4J 组件。

20.2.1　下载 Dom4J 组件

Dom4J 组件是一个非常优秀的操作 XML 文件的 Java XML API，该组件不仅性能优异、功能强大，而且还极端容易使用。同时由于该组件是一个开放源代码的软件，所以许多开源产品使用 Dom4J 组件读取 XML 文件，例如 Sun 公司的 JAXM 等。目前 Dom4J 组件比较稳定的版本为 1.6.1，具体的下载步骤如下。

（1）首先访问下载 Dom4J 组件的官方网站（http://www.dom4j.org），如图 20.9 所示。在该页面中单击 Download 链接就可以转到关于 Dom4J 组件的相关页面。

（2）在 Dom4J 组件相关页面中（如图 20.10 所示），单击 SourceForge 链接就可以转到关于该组件下载的页面。

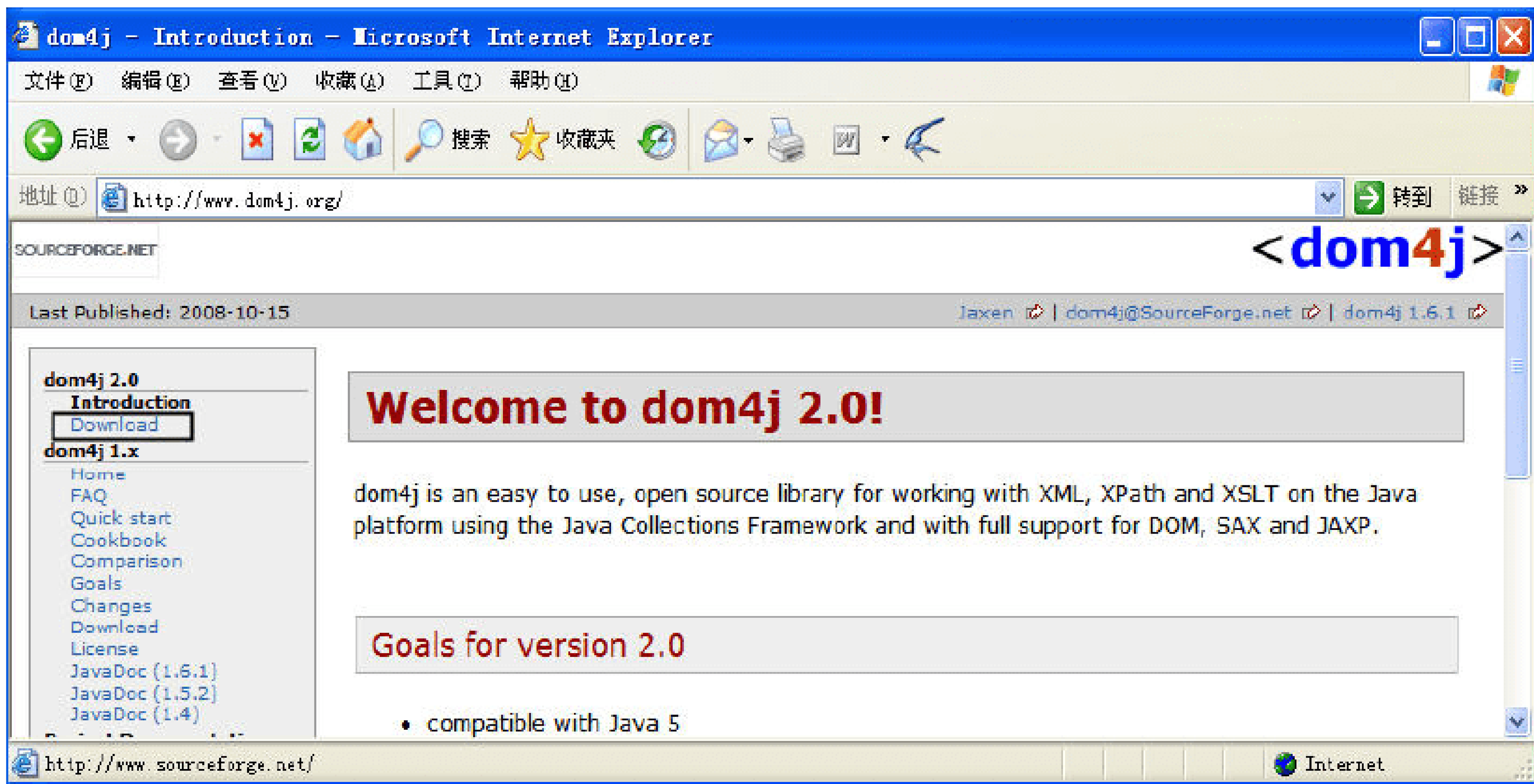


图 20.9　Dom4J 组件首页

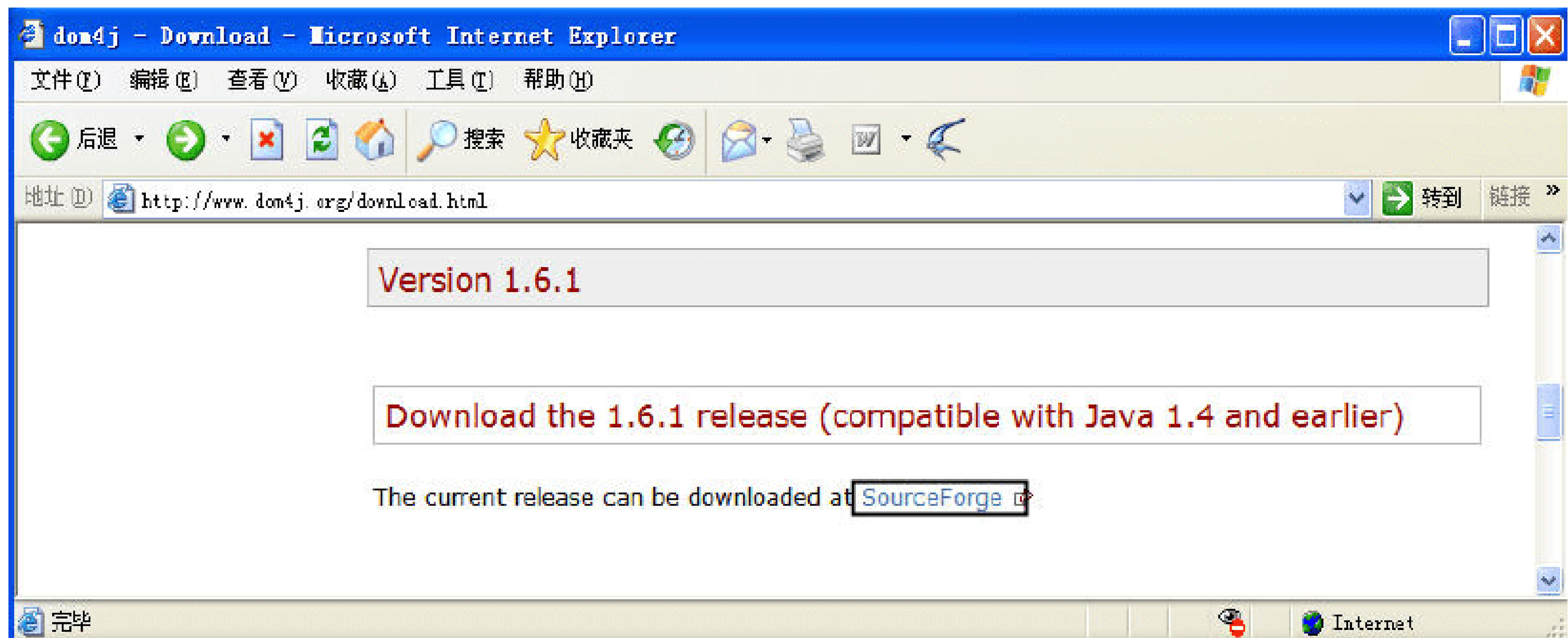


图 20.10　Dom4J 组件相关页面

（3）在关于 Dom4J 组件下载的相关页面中（如图 20.11 所示），单击 dom4j-1.6.1.zip 链接就可以实现该版本组件的下载。

Browse Files for dom4j: flexible XML framework for Java

File/Folder Name	Platform	Size	Date	Notes/Subscribe
Newest Files				
dom4j-1.6.1.zip	Platform-Independent	11.2 MiB	Mon May 16 2005 12:08	
dom4j-1.6.1.tar.gz	Platform-Independent	9.2 MiB	Mon May 16 2005 12:07	
dom4j-1.6.1.jar	Platform-Independent	306.5 KiB	Mon May 16 2005 12:07	
All Files				
dom4j				

图 20.11　实现 Dom4J 组件的下载

至此，就完成了对 Dom4J 组件的下载。

20.2.2　安装和配置 Dom4J 组件

20.2.1 节介绍了如何下载 Dom4J 组件，下载完该 jar 文件后就可以在 Java Web 项目中使用该框架。在具体使用之前，先解压 dom4j-1.6.1.zip 文件，该压缩包目录如图 20.12 所示，各个文件的作用如下。

- docs：Dom4J 组件相关 jar 文件的帮助文档；
- lib：关于 Dom4J 组件开发时的一些 jar 文件；
- src：Dom4J 组件相关 jar 文件的源代码；
- dom4j-1.6.1.jar：Dom4J 组件相关 jar 文件。

为了便于使用，可以复制 dom4j-1.6.1.jar 文件到相应的文件夹中，然后为这个文件夹在 MyEclipse 开发环境中专门建立一个名为 dom4j 的用户库，如图 20.13 所示。

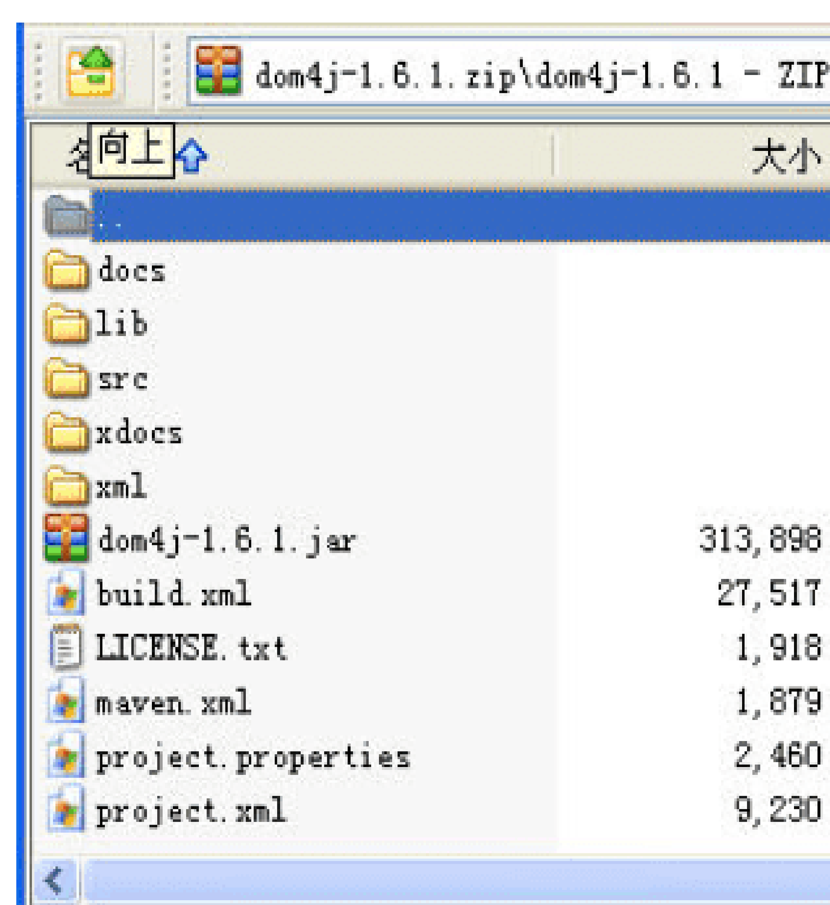


图 20.12　目录结构

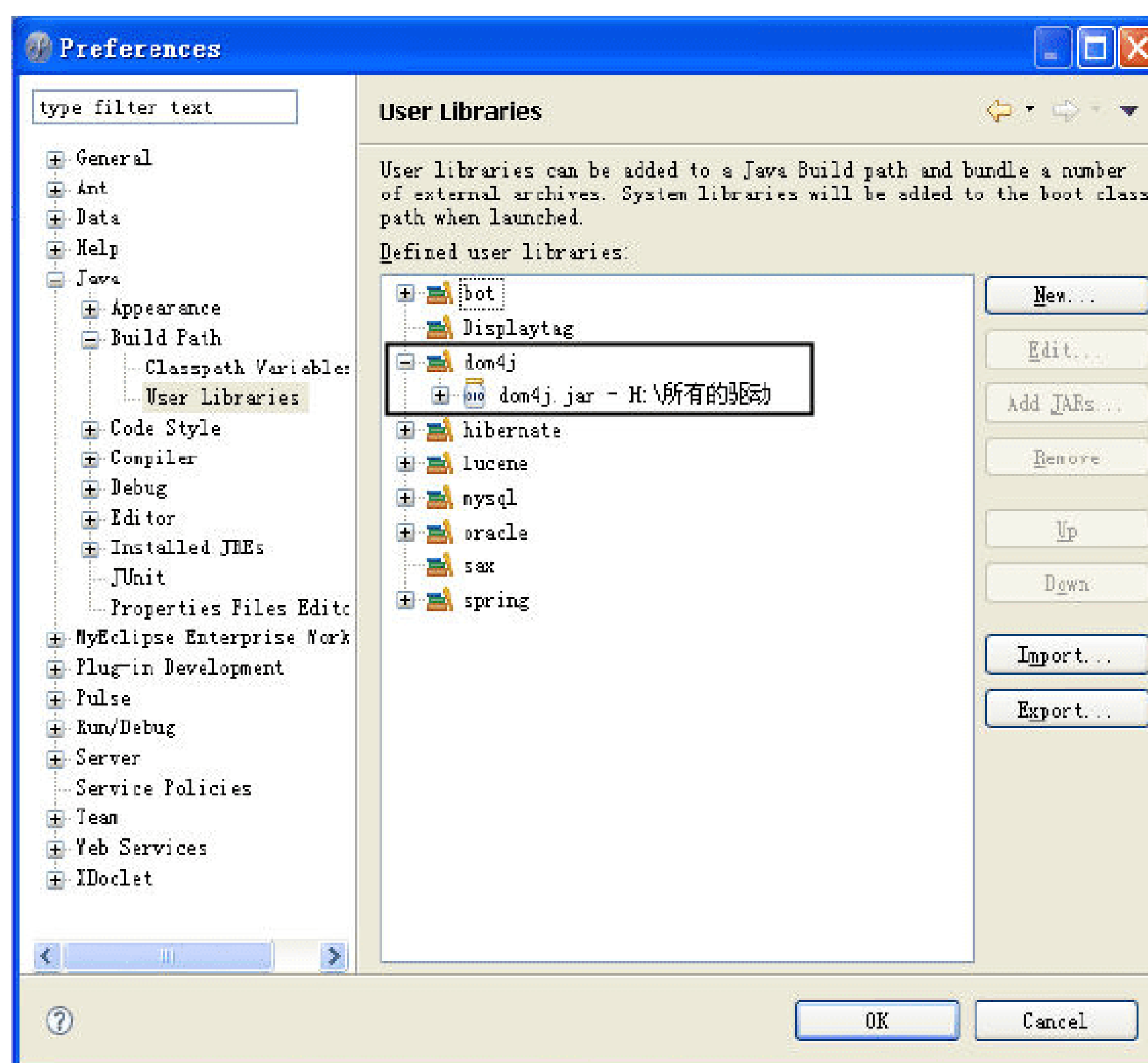


图 20.13　配置 dom4j 用户库

至此，就完成了 dom4j 用户库的配置。

20.2.3　Dom4J 组件的简单使用——解析 XML 文件

本节将通过一个简单的实例来讲解，如何通过 Dom4j 组件的 API 来实现解析 XML 文件。Dom4jApp.java 文件用来实现解析 dept.xml 文件，同时输出该 XML 文件中的内容，具体内容如代码 20.1 所示。dept.xml 文件的具体内容如代码 20.2 所示。

代码 20.1　解析 XML 文件：Dom4jApp.java

```
...
public class Dom4jApp {
    public Dom4jApp()                                   //构造函数
    {
    }
    public static void parseXML()                       //解析 XML 文件
    {
        SAXReader parser=new SAXReader();               //获取解析对象
        try{
            Document doc=parser.read(new File("depts.xml"));
                                                        //获取 Document 对象
            //获取和输出根元素
            Element root=doc.getRootElement();          //获取根元素对象
            String rootName=root.getName();             //获取根元素对象名称
```

```
            System.out.println(rootName);          //输出根元素名称
            //获取和输出儿子元素
            List<Element> list=root.elements();    //获取根元素下的儿子对象
            //遍历根元素下的儿子对象
            for(Element e:list)
            {
                String eName=e.getName();          //获取儿子对象的名字
                System.out.println(eName);         //输出儿子对象的名称
                List<Attribute> atts=e.attributes();
                                                   //获取当前儿子的属性对象
                //遍历儿子对象的属性
                for(Attribute att:atts)
                {
                    String attName=att.getName();      //获取属性的名字
                    String attValue=att.getValue();    //获取属性的值
                    System.out.println(attName+"---"+attValue);
                }
                //获取和输出孙子对象
                Iterator<Element> iter=e.elementIterator();
                                                       //获取孙子对象
                //遍历孙子对象
                while(iter.hasNext())
                {
                    Element child=iter.next();
                    String childName=child.getName();  //获取孙子对象名称
                    String childText=child.getText();  //获取孙子元素的内容
                    System.out.println(childName+"---"+childText);
                }
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    public static void main(String[] args)             //主函数
    {
        parseXML();
    }
}
```

代码 20.2　XML 文件内容：dept.xml

```
<?xml version="1.0" encoding="GBK"?>
<depts>
    <dept deptid="1">                        <!--编号为 1 的部门记录 -->
        <deptname>行政部</deptname>
        <deptnum>20</deptnum>
        <deptdesc>行政相关</deptdesc>
    </dept>
</depts>
```

运行该文件的结果如图 20.14 所示。

【代码解析】

- 首先通过 Dom4j 组件 API 中 SAXReader 类创建一个解析对象，然后通过该类的 read()方法获取所要解析 XML 文件的 Document 对象。对于 read()方法，其参数为所要解析 XML 文件的路径；

- 如果想获取关于根元素的信息，首先可以通过 Document.getRootElement()方法获取根对象，而根对象拥有一个 getName()方法用来获取根元素的名称；

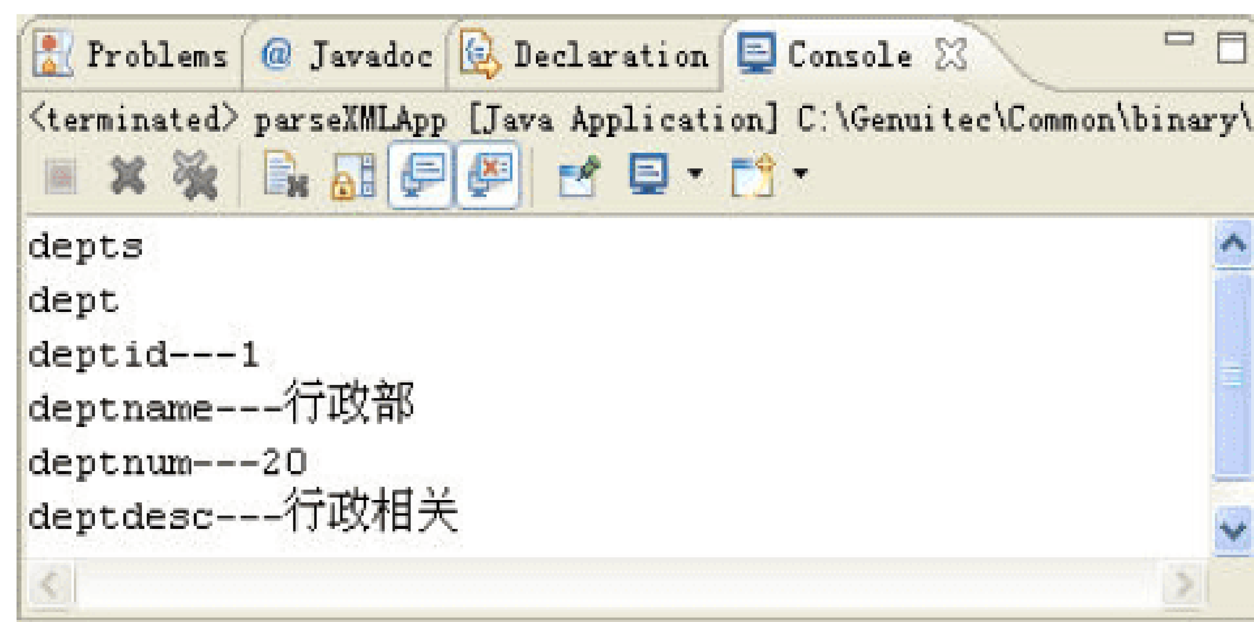


图 20.14　运行结果

- 获取根元素的相关信息后，接着就需要获取儿子元素的相关信息。通过根元素对象的 elements()方法可以获取儿子元素对象，同理儿子元素对象中也存在方法 getName()可以获取该对象的名称。由于儿子元素一般会具有属性，所以根元素对象中还存在 attribute()方法，该方法可以获取属性对象。如果想获取属性的名称和值，就需要属性对象的 getName()和 getValue()方法；
- 在儿子元素中一般会存在好多个孙子元素，通过儿子对象的 elementIterator()就可以获取孙子对象。孙子对象一般没有属性，但是却有内容，这两个对象可以通过孙子对象的 getName()和 getText()方法获取。

上述代码是针对一般 XML 文件的解析代码，比较复杂。如果只解析 dept.xml 文件，上述代码的具体内容可以修改为如代码 20.3 所示。

代码 20.3　解析 XML 文件：Dom4jApp.java

```
...
public class Dom4jApp {
    public static void parseXML()
    {
        SAXReader parser=new SAXReader();            //获取解析对象
        try{
            //关于根元素的相关信息
            Document doc=parser.read(new File("dept.xml"));
                                                     //获取 Document 对象
            Element root=doc.getRootElement();       //获取根对象
            String rootName=root.getName();          //获取根元素的名称
            System.out.println(rootName);
            //关于儿子对象的相关信息
            Element e1=root.element("dept");         //获取儿子对象
            String eName=e1.getName();               //获取儿子对象的名称
            System.out.println(eName);
            Attribute att=e1.attribute("deptid");    //得到儿子对象的属性对象
            String attName=att.getName();            //获取属性对象的名称
            String attValue=att.getValue();          //获取属性对象的值
            System.out.println(attName+"---"+attValue);
            //关于名为 deptname 孙子对象的相关信息
            Element e21=e1.element("deptname");      //获取孙子对象
            String childName21=e21.getName();        //孙子对象的名称
            String childText21=e21.getText();        //孙子对象的内容
            System.out.println(childName21+"---"+childText21);
            //关于名为 deptnum 孙子对象的相关信息
            Element e22=e1.element("deptnum");
            String childName22=e22.getName();
            String childText22=e22.getText();
            System.out.println(childName22+"---"+childText22);
            //关于名为 deptdesc 孙子对象的相关信息
```

```
            Element e23=e1.element("deptdesc");
            String childName23=e23.getName();
            String childText23=e23.getText();
            System.out.println(childName23+"---"+childText23);
        }catch(Exception e){
            e.printStackTrace();
        }
    }
...
```

20.2.4　Dom4j 组件的简单使用——创建 XML 文件

本章将通过一个简单的实例来讲解，如何通过 Dom4j 组件的 API 来创建 XML 文件。Dom4jApp.java 文件用来实现创建文件，具体内容如代码 20.4 所示。

代码 20.4　创建 XML 文件：Dom4jApp.java

```
...
public class Dom4jApp {
    public static void createXML()                  // 创建 XML 文件
    {
        DocumentFactory f = new DocumentFactory();
                                                    //创建 DocumentFactory 对象
        Document doc = f.createDocument();          //获取 Document 对象
        doc.addComment("人的信息 xml 文件");          //设置注释信息
        // 设置根元素
        Element root = doc.addElement("peoples");
                                                    //为 Document 对象设置根元素
        // 设置第一个儿子元素
        Element p1 = root.addElement("person");//为根对象设置儿子元素
        p1.addAttribute("pid", "1");                //设置儿子元素的属性
        p1.addComment("第一个人");                   //设置儿子元素的注释
        // 设置孙子元素
        Element pnameEle = p1.addElement("pname"); //为儿子对象设置孙子元素
        pnameEle.setText("张三");                         //设置孙子对象的内容
        Element psexEle = p1.addElement("psex");
        psexEle.setText("男");
        Element pageEle = p1.addElement("page");
        pageEle.setText("20");
        Element phoneEle = p1.addElement("phone");
        phoneEle.setText("13556746645");
        // 设置第二个儿子元素
        Element p2 = root.addElement("person");
        p2.addAttribute("pid", "2");
        p2.addComment("第二个人");
        // 设置孙子元素
        Element pnameEle2 = p2.addElement("pname");
        pnameEle2.setText("张三");
        Element psexEle2 = p2.addElement("psex");
        psexEle2.setText("男");
        Element pageEle2 = p2.addElement("page");
        pageEle2.setText("20");
        Element phoneEle2 = p2.addElement("phone");
        phoneEle2.setText("13556746645");
```

```
        // 设置第三个儿子元素
        Element p3 = p2.createCopy();       //通过第二个儿子复制设置第三个儿子
        p3.addComment("第三个人");
        p3.attribute("pid").setValue("3");       //修改第三个儿子的属性
        p3.element("pname").setText("李四");     //修改第三个儿子的孙子元素
        root.add(p3);                            //第三个儿子添加到根元素中
        try {
            OutputFormat format = new OutputFormat();
                                            //定义把 document 进行输入的格式
            format.setEncoding("utf-8");         //输入的编码格式
            format.setIndent(true);              //输入是否缩进
            format.setIndent("  ");              //输入缩进的间距
            format.setNewlines(true);            //换行输出
            format.setSuppressDeclaration(true);
            OutputStream os = new FileOutputStream("peoples.xml");
                                                 //获取输出流
            XMLWriter writer = new XMLWriter(os, format);
                                                 //设置输出流的格式
            writer.write(doc);                   //输出 doc 内容
            writer.close();                      //关闭资源
            os.close();                          //关闭资源
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args)       //主函数
    {
        createXML();                             //调用相应方法
        System.out.println("创建 XML 文件成功！");
    }
}
```

运行该文件的结果如图 20.15 所示。而所创建的 XML 文件的具体内容如代码 20.5 所示。

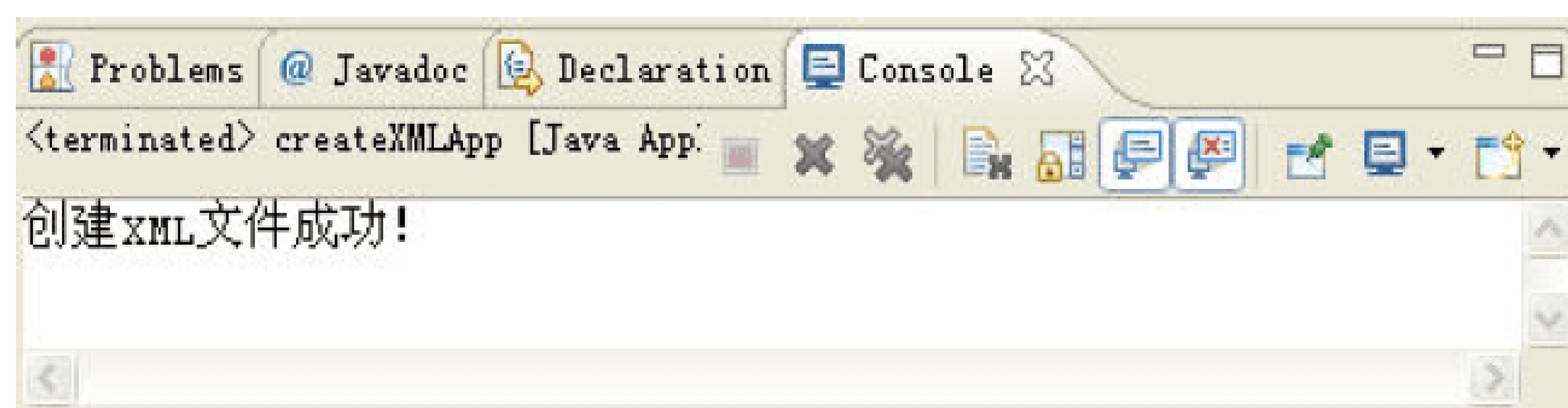


图 20.15　运行结果

代码 20.5　创建 XML 文件：peoples.xml

```
<!--人的信息 XML 文件-->
<peoples>
  <person pid="1">
    <!--第一个人-->
    <pname>张三</pname>
    <psex>男</psex>
    <page>20</page>
    <phone>13556746645</phone>
  </person>
  <person pid="2">
```

```
    <!--第二个人-->
    <pname>张三</pname>
    <psex>男</psex>
    <page>20</page>
    <phone>13556746645</phone>
  </person>
  <person pid="3">
    <!--第三个人-->
    <pname>李四</pname>
    <psex>男</psex>
    <page>20</page>
    <phone>13556746645</phone>
  </person>
</peoples>
```

【代码解析】

- 创建 XML 文件正好与解析 XML 文件相反，首先创建 Document 对象，然后按照树形结构设置该对象的各种元素；
- 如果想创建 Document 对象，可以通过工厂类 DocumentFactory 的方法 createDocument()来实现；
- 如果想设置 Document 对象的根元素，可以通过该对象的 addElement()方法来实现；
- 如果想设置根元素对象的儿子元素，可以通过该对象的 addElement()方法来实现。由于儿子元素一般需要设置属性和孙子元素，所以儿子元素对象还拥有 addAttri-hbute()和 addElement()方法；
- 如果想设置孙子元素的内容，可以通过孙子对象的 setText()方法来实现；
- 如果两个儿子元素的内容基本相同，可以通过 createCopy()方法复制一个元素，然后再获取相关元素通过相应的 setXXX()方法修改部分内容；
- 设置好 Document 对象后，就可以通过输入流把 Document 对象输入到相应的 XML 文件中。

20.3　数据格式转换功能前期准备

本节除了将详细地介绍如何设计关于数据格式转换功能的数据库和表外，还将配置实现该系统将利用的 Struts 2.x+Hibernate+MySQL 框架的环境。其中 Struts 2.x 框架的版本为 Struts 2.0、Hibernate 框架的版本为 Hibernate 3.0，以及数据库 MySQL 版本为 MySQL 5.0。

20.3.1　数据库设计

数据格式转换功能需要建立一个数据库并在该数据库中创建一张表，存放表格的数据库 dept 和存放部门信息的表 Dept。

1．创建数据库dept

如果想创建数据库 dept，可以在 MySQL 的命令行窗口中输入如下命令：

```
CREATE DATABASE 'zdept'
```

2. 创建表Dept

如果想创建表 Dept，可以在 MySQL 的命令行窗口中输入相关命令。该表格的模拟数据如图 20.16 所示。

	DEPTID	DEPTNAME	DEPTNUM	DEPTDESC
□	1	人事部门	1	管理人员
□	2	资源部门	2	分配资料
*				(NULL)

图 20.16　orderinfo 表格的模拟数据

该表的具体信息如表 20.1 所示。

表 20.1　表Dept信息

字 段 名 称	数 据 类 型	字 段 说 明
Deptid	int(11)	部门编号
Deptname	varchar(30)	部门名称
Deptnum	int(11)	部门号码
Deptdesc	Varchar(50)	部门描述

实现部门信息模型的内容如代码 20.6 所示。关于该类的映射文件内容如代码 20.7 所示。

代码 20.6　部门对象：Dept.java

```
...
public class Dept  implements java.io.Serializable {
   //创建字段
   private Long deptid;
   private String deptname;
   private Long deptnum;
   private String deptdesc;
   public Dept() {                                    //无参构造函数
   }
   public Dept(String deptname, Long deptnum) {    //有参构造函数
      this.deptname = deptname;
      this.deptnum = deptnum;
}

//省略 deptid、deptname、deptnum、deptdesc 属性的编写
...
}
```

代码 20.7　部门对象映射文件：Dept.hbm.xml

```
...
<hibernate-mapping>
   <!--制定要持久化的类与对应数据库的表映射关系-->
<class name="org.itfuture.www.po.Dept" table="DEPT" schema="ITFUTURE">
      <!--表 DEPT 主键的设置>
      <id name="deptid" type="java.lang.Long">
         <column name="DEPTID" precision="10" scale="0" />
         <generator class="assigned" />
      </id>
      <!--表 DEPT 字段 DEPTNAME 与 deptname 的映射关系>
      <property name="deptname" type="java.lang.String">
         <column name="DEPTNAME" length="20" not-null="true" />
```

```
        </property>
        <!--表 DEPT 字段 DEPTNUM 与 deptnum 的映射关系>
        <property name="deptnum" type="java.lang.Long">
            <column name="DEPTNUM" precision="8" scale="0" not-null="true"/>
        </property>
        <!--表 DEPT 字段 DEPTDESC 与 deptdesc 的映射关系>
        <property name="deptdesc" type="java.lang.String">
            <column name="DEPTDESC" length="50" />
        </property>
    </class>
</hibernate-mapping>
```

注意：Dept.java 类可以参考数据库中的表格 Dept 来编写，而该类的映射文件则可以通过向导来实现。

至此，就完成了对数据格式转换功能数据库和表的设计。

20.3.2　关于 Struts 2.x 的准备

在实现 Struts 2.x 框架与 Hibernate 框架二者集成时，对于其中的 Struts 2.x 框架除了需要引入相应的 jar 文件外，还必须得对 struts.xml 和 web.xml 文件做相应的配置。

1. 引入jar文件

首先引入关于 Struts 2.0 框架的核心包：struts2-core-2.0.11.jar、xwork-2.0.4.jar、ognl-2.6.11.jar、freemarker-2.3.8.jar 和 commons-logging-l.0.4.jar。由于需要连接数据库 MySQL，所以还需要引入关于该数据库的驱动 mysql-connector-java-3.1.14-bin.jar。

注意：前 6 个 jar 包在 Struts 2.0 框架的 jar 文件中就可以找到，而最后一个关于数据库的驱动则必须从该数据库的官方网站上下载。

2. 修改web.xml文件

为了使数据格式转换功能支持 Struts 2.0 框架，需要在 web.xml 文件中增加如代码 20.8 所示的内容。

代码 20.8　修改 web.xml 文件：web.xml

```
...
    <!--设置过滤器类-->
    <filter>
        <filter-name>s</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <!--设置过滤器映射-->
    <filter-mapping>
        <filter-name>s</filter-name>
        <url-pattern>/*</url-pattern>
```

```
    </filter-mapping>
</web-app>
```

3. 创建struts.xml文件

为了使数据格式转换功能支持 Struts 2.0 框架，还需要在 ReportJxl/src 目录下创建 struts.xml 文件，该文件的内容如代码 20.9 所示。

代码 20.9　实现 struts.xml 文件：struts.xml

```
...
<struts>
  <constant name="struts.i18n.encoding" value="GBK"></constant>
  <constant name="struts.action.extension" value="action,do,itfuture">
  </constant>
  <constant name="struts.locale" value="zh_CN"></constant>
  <constant name="struts.devMode" value="true"></constant>
  <constant name="struts.enable.SlashesInActionNames" value="true">
  </constant>
  <!--引入外部配置文件 -->
  <include file="actions.xml"></include>
</struts>
```

至此，就完成了对数据格式转换功能中的 Struts 2.0 框架的配置。

20.3.3　关于 Hibernate 3.0 的准备

在实现 Struts 2.x 框架与 Hibernate 3.0 二者集成时，对于其中的 Hibernate 3.0 框架除了需要引入相应的 jar 文件外，还必须对 hibernate.cfg.xml 文件做相应的配置。

1. 引入jar文件

首先引入关于 Hibernate 3.0 框架的核心包：Hibernate3.0.jar、log4j-1.2.13.jar、cglib-nodep-2.1_3.jar、dom4j-1.6.1.jar、commons-collections.jar、c3p0-0.9.0.4.jar、jta.jar 和 antlr-2.7.6.jar。

2. 创建hibernate.cfg.xml文件

首先通过 MyEclipse 开发环境中关于 Hibernate 框架的向导，让数据格式转换功能支持 Hibernate 3.0 框架，然后通过该开发环境的反向工程向导，实现对数据库 dept 中的表 Dept 进行关系数据库到对象的映射。hibernate.cfg.xml 文件的内容如代码 20.10 所示。

代码 20.10　实现 hibernate.cfg.xml 文件：hibernate.cfg.xml

```
...
    <session-factory>
        <!-- 指定连接数据库用户名-->
        <property name="connection.username">root</property>
        <!-- 指定连接数据库 URL -->
        <property name="connection.url">
            votesystem:mysql://hostname/dept
        </property>
```

```
        <!-- 指定连接数据库方言-->
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <!-- 指定连接数据库用户名-->
        <property name="myeclipse.connection.profile">mysql</property>
        <!-- 指定连接数据库密码-->
        <property name="connection.password">root</property>
        <!-- 指定连接数据库驱动-->
        <property name="connection.driver class">
            com.mysql.votesystem.Driver
        </property>
        <!-- 指定 Hibernate 映射文件 -->
        <mapping resource="org/itfuture/www/po/Dept.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

至此，就完成了对数据格式转换功能中 Hibernate 3.0 框架的配置。

20.4　数据格式转换功能具体开发

为了让读者可以快速地理解和掌握数据格式转换功能，在具体讲解时对该功能按照 MVC 模式分成为领域模型层、持久层、业务层和表现层。由于在前面章节中已经介绍了领域模型层，所以本节将介绍关于该项目的其他层。

20.4.1　关于 Dept 表的持久层

在实现关于 Dept 表操作时，采用了 DAO 模式。本节将实现对表 Dept 操作：每页的分页记录方法、分页的总页数方法、保存部门记录方法和获取一条部门记录方法。实现操作 Dept 表的接口如代码 20.11 所示。

代码 20.11　操作 Dept 表：DeptDao.java

```
...
public interface DeptDao {
   List getDepts(int curpage,int pagerecord,String cond);
                                                 //关于每页的分页记录
   int count(String cond);                       //关于分页的总页数
   void save(Dept dept) throws Exception;        //保存部门记录
   Dept getDept(Long deptid);                    //获取一条部门记录
}
...
```

【代码解析】

- 上述代码中 getDepts()和 count()方法用来为实现分页做准备；
- 上述代码中 getDept()方法主要用来为数据库数据记录转换成 XML 文件数据做准备，而 getDept()方法则用来为 XML 文件数据转换成数据库数据记录做准备。

实现关于部门表 Dept 接口 DeptDaoi 的类如代码 20.12 所示。

代码 20.12　操作 Dept 表：DeptDaoi.java

```
...
public class DeptDaoi extends HibernateDaoSupport implements DeptDao {
    public DeptDaoi() {                                    //构造函数
    }
    public int count(String cond) {                        //获取分页总数
        return cond;
    }
    public Dept getDept(Long deptid)                       //获取一个部门记录
    {
        return (Dept)this.getHibernate_Template().get(Dept.class, deptid);
    }
    public List getDepts(int curpage, int pagerecord, String cond)
                                                           //为分页做准备
    {
        String hql="from Dept A where 1=1 "+cond;
        return this.getHibernate_Template().getObjects(hql, curpage,
        pagerecord);
    }
    public void save(Dept dept) throws Exception           //实现保存部门记录
    {
        this.getHibernate Template().save(dept);
    }
}
...
```

注意：在上述代码中，分别实现了 DeptDao 接口中的所有方法。

20.4.2　关于数据格式转换服务层

在实现关于数据格式转换服务层时，采用了 DAO 模式。本节将实现服务层方法：获取每条部门记录方法、实现从 XML 文件向数据库记录转换方法，以及实现从数据库记录向 XML 文件数据转换方法。

DeptService.java 文件主要实现了数据库记录与 XML 文件数据相互转换的方法，该文件的具体内容如代码 20.13 所示。

代码 20.13　创建数据存储转换：DeptService.java

```
...
public interface DeptService {
   int pagerecord=5;                                   //记录每页的记录数
   Pageinfo getDepts(int curpage,String cond);         //获取一条部门记录
   void FromXmlToDB(HttpServletRequest request,String fileName);
                                                       //XML 数据转成数据库数据
   void FromDBToXml(Long deptid,HttpServletRequest request);
                                                       //数据库数据转成 XML 数据
}
...
```

实现关于数据格式转换功能接口 DeptDaoi 的类如代码 20.14 所示。

代码 20.14　实现数据存储转换：DeptServicei.java

```
...
public class DeptServicei extends HibernateDaoSupport implements
DeptService {
    private DeptDao dao;                                    //创建字段 dao
    public DeptDao getDao()                                 //配置属性 dao
    {
        return dao;
    }
    public void setDao(DeptDao dao)
    {
        this.dao = dao;
    }
    public DeptServicei()                                   //构造函数
    {
        this.setDao(new DeptDaoi());
    }
    //实现数据库数据转换成 XML 数据
    public void FromDBToXml(Long deptid, HttpServletRequest request)
    {
        Dept dept=this.getDao().getDept(deptid);            //获取一条部门记录
        try{
            DocumentFactory f=new DocumentFactory();
            Document doc=f.createDocument();                //获取 Document 对象
            Element root=doc.addElement("depts");           //设置根元素
            Element e=root.addElement("dept");              //设置儿子元素
            e.addAttribute("deptid",dept.getDeptid().toString());
            Element deptnameEle=e.addElement("deptname");//设置孙子元素
            deptnameEle.setText(dept.getDeptname());        //设置孙子元素内容
            Element deptnumEle=e.addElement("deptnum");
            deptnumEle.setText(dept.getDeptnum().toString());
            Element deptdescEle=e.addElement("deptdesc");
            deptdescEle.setText(dept.getDeptdesc());
            //获取 XML 文件的路径
            String path=request.getSession().getServletContext().
            getRealPath("/xml");
            File folder=new File(path);
            if(!folder.exists())
                folder.mkdir();
            OutputStream os=new FileOutputStream(path+"/dept"+deptid+
            ".xml");
            //创建格式
            OutputFormat format=new OutputFormat();
            format.setEncoding("GBK");
            format.setIndent(true);
            format.setIndent("  ");
            format.setNewlines(true);
            XMLWriter writer=new XMLWriter(os,format);
            writer.write(doc);
            writer.close();
            os.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    //实现 XML 数据转换成数据库数据
    public void FromXmlToDB(HttpServletRequest request, String fileName)
```

```
    {
      this.getHibernate_Template().beginTran();    //开启事务
      //获取 XML 文件的路径
      String allpath=request.getSession().getServletContext().getReal-
      Path("/xml");
      allpath=allpath+"/"+fileName;                //得到完整的路径
      SAXReader reader=new SAXReader();             //获取 SAXReader 对象
      try{
        Document doc=reader.read(allpath);          //获取 Document 对象
        Element root=doc.getRootElement();          //获取 Element 元素
        List<Element> list=root.elements();
        for(Element e:list)                         //遍历 Element 元素
        {
            Dept dept=new Dept();                   //创建 Dept 对象
            Attribute att=e.attribute(0);
            String attName=att.getName();
            String attValue=att.getValue();
            BeanUtils.setProperty(dept,attName,attValue);
                                            //存储 Element 元素到 Dept 对象
            List<Element> children=e.elements();
            for(Element child:children)             //遍历 children 元素
            {
                String eName=child.getName();
                String eValue=child.getTextTrim();
                BeanUtils.setProperty(dept,eName,eValue);
            }
            this.getDao().save(dept);                   //保存到数据库
        }
        this.getHibernate_Template().commitTran();      //提交事务
        }catch(Exception e){
        e.printStackTrace();
        this.getHibernate Template().rollbackTran();    //回滚事务
        }
        this.getHibernate_Template().closeSession();    //关闭 Session 对象
    }
    public Pageinfo getDepts(int curpage, String cond) //实现分页功能
    {
        List list=this.getDao().getDepts(curpage, pagerecord, cond);
                                                    //获取当前页面的所有记录
        int count=this.getDao().count(cond);            //获取所有的记录数
        Pageinfo pageinfo=new Pageinfo(curpage,count,pagerecord,list);
                                                        //生成 pageinfo 对象
        this.getHibernate_Template().closeSession();    //关闭 Session 对象
        return pageinfo;                                //返回 pageinfo 对象
    }
}
...
```

【代码解析】

- 在上述代码中实现数据库数据向 XML 数据转换的 FromXmlToDB()方法中，首先获取数据库中的数据然后做为创建 XML 文件的内容；
- 在上述代码中实现 XML 数据向数据库数据转换的 FromDBToXml()方法中，首先解析 XML 文件的内容，然后把相应的内容存储到数据库中。

20.4.3　实现页面跳转 Action 类

数据格式转换功能系统中的所有请求都由 Struts 2.0 框架中名为 DeptAction 的 Action 类来处理，DeptAction.java 的具体内容如代码 20.15 所示。

代码 20.15　数据存储转换 Action：DeptAction.java

```
...
public class DeptAction extends CommonAction {
    //创建属性
    private Pageinfo pageinfo;
    private DeptService service;
    private Dept dept;
    private File xmlfile;
    private String xmlfileFileName;
    public DeptAction()                                 //构造函数
    {
        this.setService(new DeptServicei());
    }
    public String list()                                //实现 list()方法
    {
        this.pageinfo=this.getService().getDepts(1,"");
        return "success";
    }
    public String xmltodb()                             //实现 xmltodb()方法
    {
        //获取存储 XML 文件的路径
        String path=this.getRequest().getSession().getServletContext().
        getRealPath("/xml");          File folder=new File(path);
                                                        //File 对象文件
        if(!folder.exists())
        folder.mkdir();                                 //创建的文件夹
        File file=new File(path+"/"+xmlfileFileName);  //新的空文件
        xmlfile.renameTo(file);                         //修改文件的名字
        // 实现 XML 格式向数据库格式转换
        this.getService().FromXmlToDB(this.getRequest(),xmlfileFileName);
        return this.list();
    }
    public String dbtoxml()                     //实现数据库格式转向 XML 格式
    {
        this.getService().FromDBToXml(dept.getDeptid(),this.getRequest());
        return this.list();
    }
    //省略属性 pgaeinfo、service、dept、xmlfile、xmlfilefileName
...
}
```

在具体编写该程序时，不是继承了 ActionSupport 类而是 CommonAction 类。该类不仅继承了 ActionSupport 类，而且还实现了对服务层方面类的操作，具体内容如代码 20.16 所示。

代码 20.16　配置文件：CommonAction.java

```
...
```

```
public class CommonAction extends ActionSupport implements ServletRequest
Aware,
        ServletResponseAware, RequestAware, SessionAware {
    private HttpServletRequest request;            //request 变量
    private HttpServletResponse response;          //response 变量
    private HttpSession session;                   //Session 变量
    private Map<String, String> requestMap;        //作用域
    private Map<String, String> sessionMap;        //作用域
    public CommonAction() {                        //构造函数
    }
    /*
     * 实现了 ServletResponseAware 接口中的方法，这样可以达到使用 IOC(Inversion
     of Control
     * 控制反转)，调用该方法的权力交给 struts2 框架
     */
    public void setServletResponse(HttpServletResponse arg0) {
        this.response = arg0;// response 赋值
    }
    public HttpServletResponse getResponse() {
        return response;
    }
    /*
     * 实现了接口 ServletRequestAware 中的方法，这样可以达到使用 IOC（控制反转），
     该方法的调用权力交给 struts2 框架
     */
    public void setServletRequest(HttpServletRequest arg0) {
        this.request = arg0;                    // request 赋值
        this.session = request.getSession();    // 由 request 得到 Session
    }
    public HttpServletRequest getRequest() {
        return request;
    }
    /*
     * 实现了 SessionAware 接口中的方法，这样可以达到使用 IOC（控制反转），该方法的调
     用权力交给 struts2 框架
     */
    public void setSession(Map arg0) {
        this.sessionMap = arg0;
    }
    public HttpSession getSession() {
        return session;
    }
    /*
     * 实现了 RequestAware 接口中的方法，这样可以达到使用 IOC（控制反转），该方法的调
     用权力交给 struts2 框架
     */
    public void setRequest(Map arg0) {
        this.requestMap = arg0;
    }
    public Map getRequestMap() {
        return requestMap;
    }
}
```

在上述代码中通过 Struts 2.0 提供的 IOC 的方式来实现 request、response 和 session 对象。接着在 actions.xml 文件中实现对 Action 类的配置，具体内容如代码 20.17 所示。

代码 20.17　配置文件：actions.xml

```
...
<struts>
  <!-- 创建自己的包，该包必须继承默认包-->
  <package name="admin" namespace="/admin" extends="struts-default">
     <!-- 配置 Action-->
     <action name="dept!*" method="{1}" class="org.itfuture.www.
     action.DeptAction">
       <result name="success">/dept.jsp</result>
     </action>
  </package>
</struts>
```

【代码解析】

在上述代码中，当关于 dept 请求处理成功后，就会转到显示部门列表的页面 dept.jsp。在配置<action>标签的 method 属性时，可以通过传递的参数来决定调用的是 xmltodb()方法还是 dbtoxml()方法。

对于 Struts 2.0 框架的配置可以在另一个 XML 文件中实现，然后把 actions.xml 文件包含到该文件中，具体内容如代码 20.18 所示。

代码 20.18　配置文件：actions.xml

```
...
<struts>
    <constant name="struts.i18n.encoding" value="GBK"></constant>
    <constant name="struts.action.extension" value="action,do,itfuture">
    </constant>
    <constant name="struts.locale" value="zh CN"></constant>
    <constant name="struts.devMode" value="true"></constant>
    <constant name="struts.enable.SlashesInActionNames" value="true">
    </constant>
    <include file="actions.xml"></include>     <!--引入另一个XML配置文件-->
</struts>
```

20.4.4　关于数据格式转换功能的页面

通过 actions.xml 文件的相关配置可以发现，关于数据格式转换功能需要两个页面：dept.jsp 和 loadxml.jsp。

1．显示部门页面

dept.jsp 页面作为数据格式转换功能系统的首页，用来显示关于部门记录页面，该页面的具体内容如代码 20.19 所示。

代码 20.19　显示部门记录页面：dept.jsp

```
...
  <body>
    <table border="1" width="80%" align="center">
      <!--表格的记录标题-->
      <tr>
```

```
      <td>部门编号</td><td>部门名称</td><td>部门编制</td><td>部门描述
</td><td>操作</td>
     </tr>
     <!--遍历变量-->
     <s:iterator value="pageinfo.pagedata">
       <tr>
         <td>${deptid}</td>                    <!--显示部门 Id 号-->
         <td>${deptname}</td>                  <!--显示部门名称-->
         <td>${deptnum}</td>                   <!--显示部门编号-->
         <td>${deptdesc}</td>                  <!--显示部门描述->
         <!--链接-->
         <td><a href="./dept!dbtoxml.do?dept.deptid=${deptid}">产生 xml
         </a></td>
       </tr>
     </s:iterator>
   </table>
   <!--链接-->
   <p align="right"><a href="../loadxml.jsp">xmlTODB</a></p>
  </body>
...
```

【代码解析】

在上述代码中，当单击“产生 xml”链接时，就会发出生成 XML 文件的请求。在链接之前，会通过循环遍历输出部门的各个属性。

2．上传XML文件页面

loadxml.jsp 页面用来实现 XML 文件的上传，在该页面中用户需要选择相应的 XML 文件，该页面的具体内容如代码 20.20 所示。

代码 20.20　上传 XML 文件：loadxml.jsp

```
...
  <body>
   <!--表单 -->
   <form action="./admin/dept!xmltodb.do" method="post" enctype=
   "multipart/form-data">
     <table>
       <tr>
          <!--选择所要上传文件-->
          <td>xml 文件</td><td><input type="file" name="xmlfile"></td>
       </tr>
     </table>
     <!--提交按钮-->
     <p align="center"><input type="submit" value="提交"/></p>
   </form>
  </body>
...
```

【代码解析】

在上述代码中，<form>标签的 action 属性值为“./admin/dept!xmltodb.do”，即该表单的请求会被名为 dept!xmltodb.do 的 Action 来处理。

20.4.5　实现生成报表工具类

在数据格式转换系统中需要 3 类工具类：封装 Hibernate 框架的工具类、实现记录分页的工具类和封装对 JavaBean 操作的工具类。由于前两类在前面章节已经讲解，所以本节只讲解如何通过反射机制实现对 JavaBean 的操作，具体内容如代码 20.21 所示。

代码 20.21　操作 JavaBean：Bean_Utils.java.java

```
...
import java.lang.reflect.*;                               //引入发射包
public class Bean_Utils {
    public Bean_Utils() {                                 //构造函数
    }
    public static Object getProperty(Object obj, String property) {
                                                          //获取 JavaBean 的属性
        Class c = obj.getClass();                         //创建 Class 对象
        Object value = null;                              //创建 Object 对象
        String mname = "get" + property.substring(0, 1).toUpperCase()
                + property.substring(1);
        try {
            Method m = c.getDeclaredMethod(mname, null);
            value = m.invoke(obj, null);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return value;
    }
}
```

20.5　多学两招——其他操作 XML 文件组件

在使用 Dom4j 组件解析 XML 文件时，首先需要把关于 XML 文件的 DOM 树型结构存放在内存中，然后才能实现对该 XML 文件的解析。该种方式降低了处理大型 XML 文件的性能。

为了解决上述问题，可以使用基于事件驱动的 SAX 组件。

20.5.1　下载 SAX 类库

SAX 全称 Simple API for XML，是由 David Megginson 采用 Java 语言开发操作的 XML 文件的 API。目前 SAX 组件的最新版本为 2.0.3，具体下载步骤如下。

（1）首先访问下载 SAX 组件相关 jar 文件的官方网站（http://www.sax.sourceforge.net），如图 20.17 所示。在该页面中单击 download area 链接就可以转到 Sax 组件的相关页面。

（2）在 SAX 组件相关页面中（如图 20.18 所示），单击 sax2r3.zip 链接就可以实现该组件的下载。

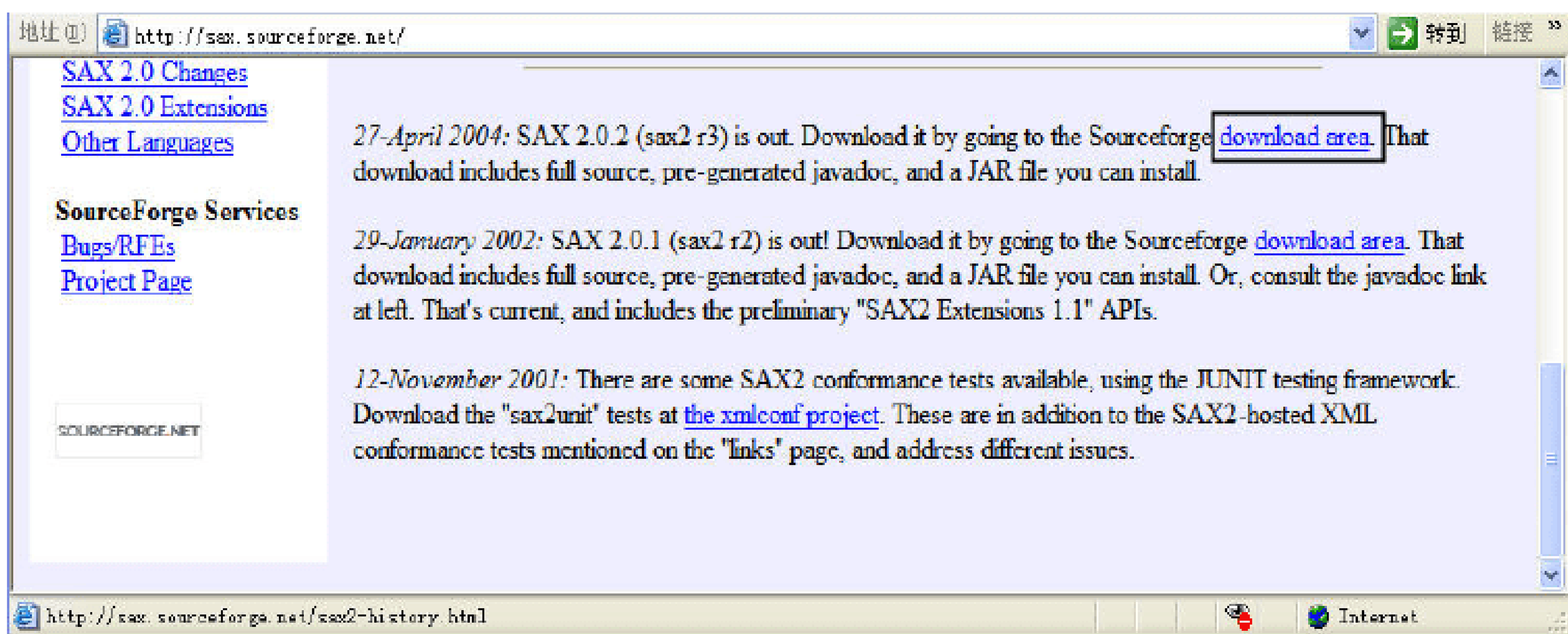


图 20.17　SAX 组件首页



图 20.18　SAX 组件的下载

至此，就完成了对 SAX 组件的下载。

20.5.2　安装和配置 SAX 组件

20.5.1 节介绍了如何下载 SAX 组件，下载完该组件后就可以在 Java Web 项目中使用该框架。在具体使用之前，先解压 sax2r3.zip 文件，该压缩包目录如图 20.19 所示。各个文件的作用如下。

- src：SAX 组件相关 jar 文件的源代码；
- docs：SAX 组件相关 jar 文件的帮助文档；
- classes：SAX 组件的类；
- sax2.jar：SAX 组件的核心 jar 文件。

为了便于使用，可以复制 sax2.jar 文件到相应的文件夹中，然后为这个文件在 MyEclipse 开发环境中专门建立一个名为 sax 的用户库，如图 20.20 所示。

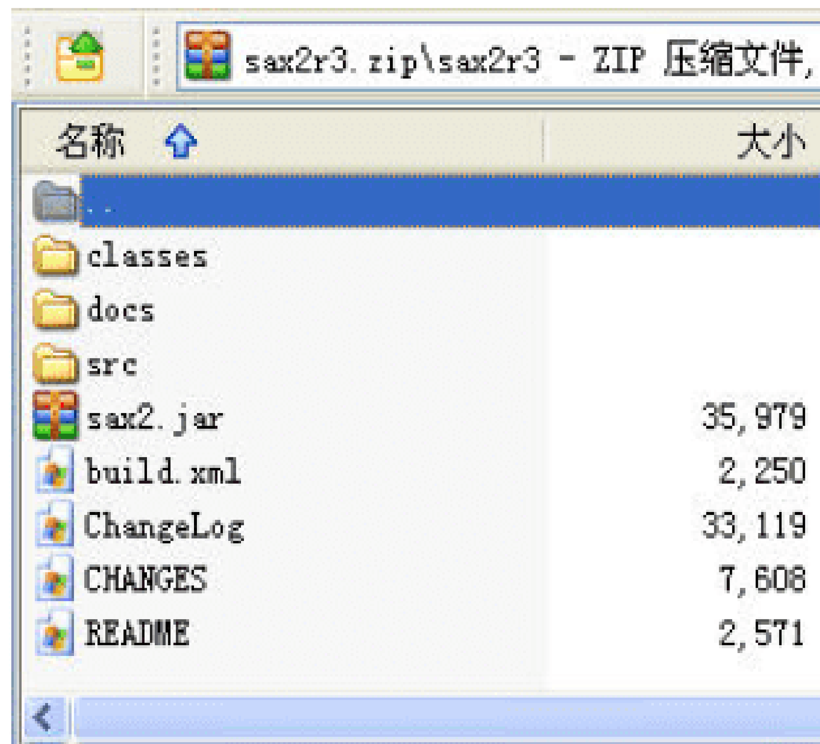


图 20.19　目录结构

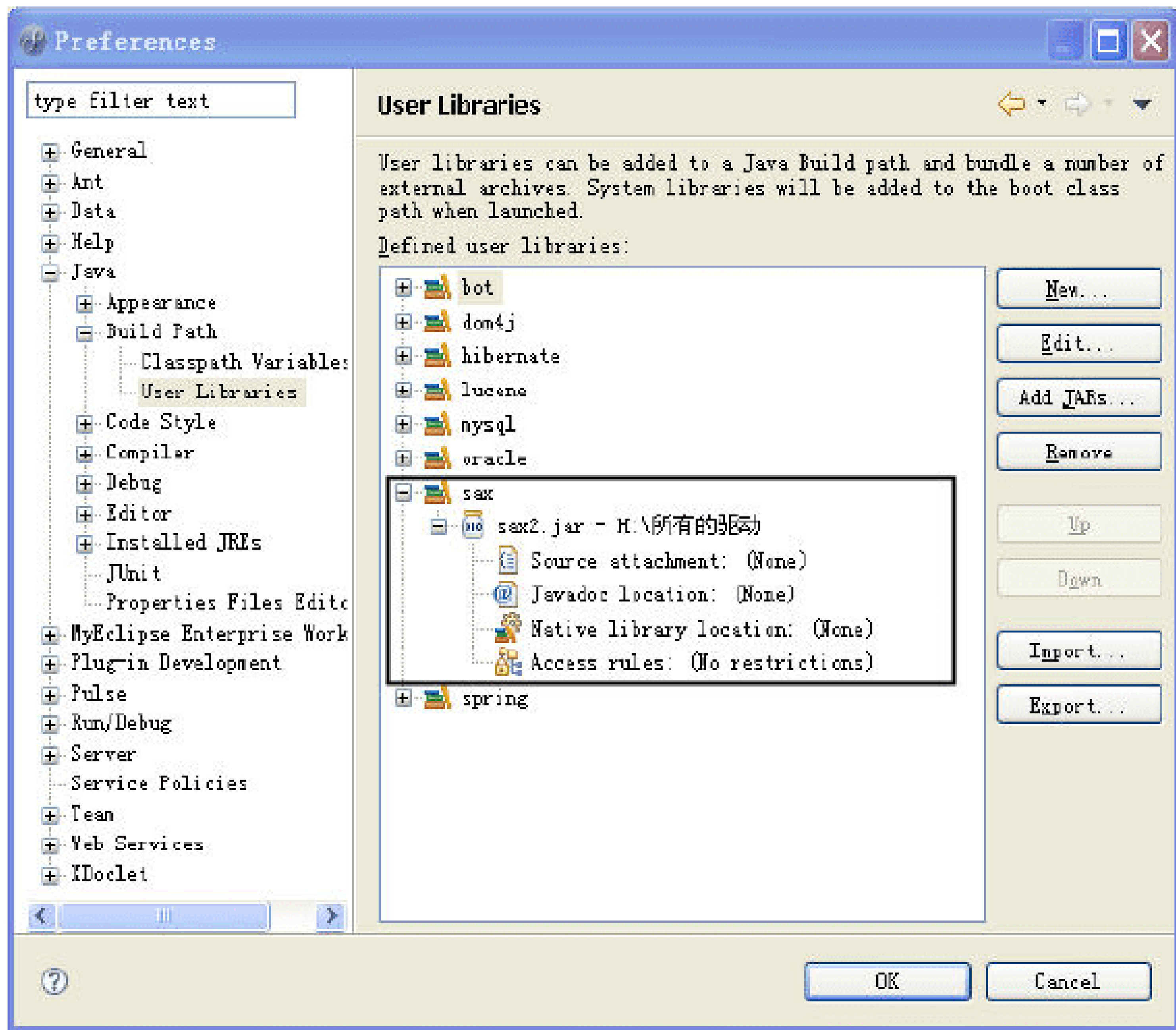


图 20.20　配置 sax 用户库

至此，就完成了对 sax 用户库的配置。

20.5.3　SAX 组件的简单使用——解析 XML 文件

为了让读者对 SAX 组件有一个大致的了解，本章将实现 SAX 组件的简单运用——对 XML 文件的解析。由于 SAX 组件是基于事件的运行机制，所以在具体编写时与 Dom4j 组件完全不同。

dept.xml 文件为所要解析的 XML 文件，具体内容如代码 20.22 所示。

代码 20.22　部门记录：dept.xml

```
<?xml version="1.0" encoding="GBK"?>
<depts>                                          <!--所有部门记录-->
    <dept deptid="1">                            <!--编号为 1 的部门-->
        <deptname>行政部</deptname>               <!--部门的名称-->
        <deptnum>20</deptnum>                    <!--部门的号码-->
        <deptdesc>行政相关</deptdesc>             <!--部门的描述-->
    </dept>
    <dept deptid="2">                            <!--编号为 2 的部门-->
        <deptname>人事部</deptname>
        <deptnum>3</deptnum>
        <deptdesc>人事相关</deptdesc>
    </dept>
</depts>
```

为了便于编写，在该项目中还专门建立一个实体对象表示部门记录对象，具体内容如代码 20.23 所示。

代码 20.23　部门记录对象：Dept.java

```
...
public class Dept
{
    //创建各种字段
    private Integer deptid;
    private String deptname;
    private Integer deptnum;
    private String deptdesc;
    public Dept()                                  //构造函数
    {
    }
    //省略 deptid、deptname、deptnum 和 deptdesc 字段属性的配置
...
    public String toString() {                     //重写 String()方法
        return this.deptid+"---"+this.deptname+"--"+this.deptnum+
        "--"+this.deptdesc+"---";
    }
}
```

注意：在上述代码中，为了便于输出，所以重写了 toString()方法。

DeptHandler.java 文件用来解析 XML 文件，具体内容如代码 20.24 所示。

代码 20.24　解析 XML 文件：DeptHandler.java

```
...
public class DeptHandler extends DefaultHandler{
    public DeptHandler()                                    //构造函数
    {
    }
    public void startDocument() throws SAXException  //拦截是否开始解析 XML
    {
        System.out.println("开始解析 xml 文档");
    }
    //拦截是否解析 XML 中某个元素的开始标记
    public void startElement(String uri, String localName, String qName,
    Attributes attributes) throws SAXException
    {
        System.out.println("uri:"+uri);
        System.out.println("localName:"+localName+"开始解析了");
        System.out.println("qName:"+qName);
        if(attributes.getLength()>0)
        {
        for(int i=0;i<attributes.getLength();i++)
        {
            String AttName=attributes.getQName(i);
            String AttValue=attributes.getValue(i);
            System.out.println(AttName+"---"+AttValue);
        }
        }
    }
    //拦截是否解析 XML 中某个元素的开始标记和结束标记中间的内容
    public void characters(char[] ch, int start, int length) throws
    SAXException
    {
        System.out.println("==="+new String(ch)+"====");
        System.out.println(new String(ch,start,length));
    }
    //拦截是否解析 XML 中某个元素的结束标记
    public void endElement(String uri, String localName, String qName)
    throws SAXException
    {
            System.out.println("uri:"+uri);
            System.out.println("localName:"+localName+"结束解析了");
            System.out.println("qName:"+qName);
    }
    public void endDocument() throws SAXException  //拦截是否结束解析 XML
    {
            System.out.println("结束解析 xml 文档");
    }
    public static void main(String[] args)          //主函数
    {
        try{
            //通过工厂生成一个解析器
            XMLReader parser=XMLReaderFactory.createXMLReader();
            //给该解析器设定句柄,对解析的事件进行拦截
            parser.setContentHandler(new DeptHandler());
            parser.parse(new InputSource("dept.xml"));      //解析器解析
        }catch(Exception e) {
```

```
            e.printStackTrace();
        }
    }
}
```

编译和运行该程序后，其运行结果如图 20.21 所示。



图 20.21　运行结果

【代码解析】

（1）SAX 组件是基于事件的 XML 组件，即在解析 XML 时会发生相应的事件，而句柄拦截器则会拦截到相应事件，进而在拦截事件的方法中获取结果，以达到解析的目的。具体流程如图 20.22 所示。

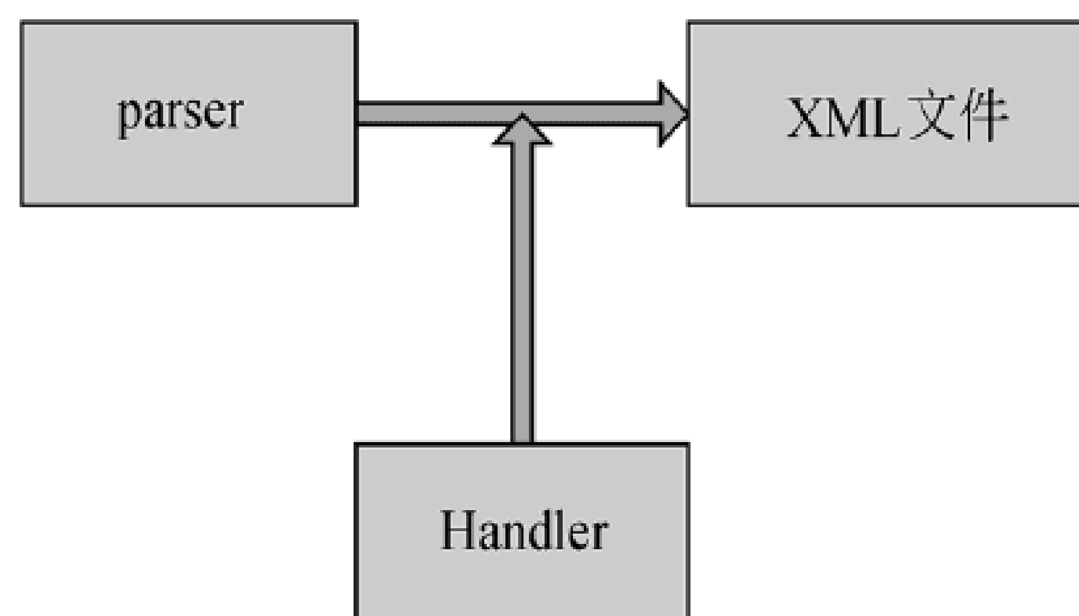


图 20.22　Sax 组件流程

（2）当 XML 文件被 XMLReader 解析器解析时会发生如下事件。

- startDoucument 事件：该方法在开始解析 XML 文档时发生；
- startElement 事件：该方法在开始解析 XML 文档的开始元素标记时发生；
- characters 事件：该方法在开始解析 XML 文档的开始元素和结束元素标记的内容时发生；
- endElement 事件：该方法在开始解析 XML 文档的结束元素标记时发生；
- endDoucument 事件：该方法在结束解析 XML 文档时发生。

（3）当发生各种解析事件时，会使用 XMLReader 解析器设定的拦截器（handler）拦截相应的事件，进而在相应的事件中得到解析的结果。

（4）在 startElement（String uri, String localName, String qName, Attributes attributes）解析开始元素标记方法中，参数 uri 表示该元素的命名空间，参数 localName 表示该元素的名称，参数 qName 表示该元素的属性名，以及参数 attributes 表示该元素的属性值。

（5）在 characters(char[] ch, int start, int length)方法中之所以会出现这 3 个参数，主要是因为需要通过 String(ch,start,length)方法输出相应的内容。

20.6　小　　结

本章主要介绍数据格式转换功能，该模块基于 Struts 2.0+Hibernate 3.0+JXL 解决方案，保持了良好的 Java EE 中 MVC 分层思想。数据格式转换项目在业务上主要分成两部分：数据库格式转换成 XML 文件格式和 XML 文件格式转换成数据库格式。在具体实现这两部分功能时，主要通过 Dom4j 组件来实现解析 XML 文件和创建 XML 文件。本章除了讲解数据转换项目外，还详细讲解了 Dom4j 组件和 SAX 组件的详细使用。

第 21 章　用户维护功能（Struts 2.x+iBATIS）

对于任何网络系统，为了能够提高自身的性能，都在想尽一切办法提高与数据库的交换速度。于是编写一个性能优秀的维护数据库系统是每个程序员追求的目标。本章将通过一个具体的案例用户维护功能，深入学习如何利用 iBATIS 框架实现关于数据库的操作。

本章将通过 Struts 2.x+iBATIS 框架技术来实现用户维护功能，其中利用 iBATIS 框架实现数据库的连接，以及利用 Struts 2.x 框架来实现表示层。

21.1　用户维护功能

用户维护的功能实际上就是在数据库中实现增加记录、删除记录和修改记录等操作。虽然实现的功能非常简单，但是为了能够提高与数据库的交换速度，在本章的用户维护模块中使用 iBATIS 框架来实现连接数据库。

21.1.1　用户维护模块结构框架分析

对于一个大型网上系统来说，实现一个可用的用户维护模块要考虑的情况十分复杂，例如用户维护页面如何实现人性化？如何设计数据库字段等。本系统的目录如图 21.1 所示。

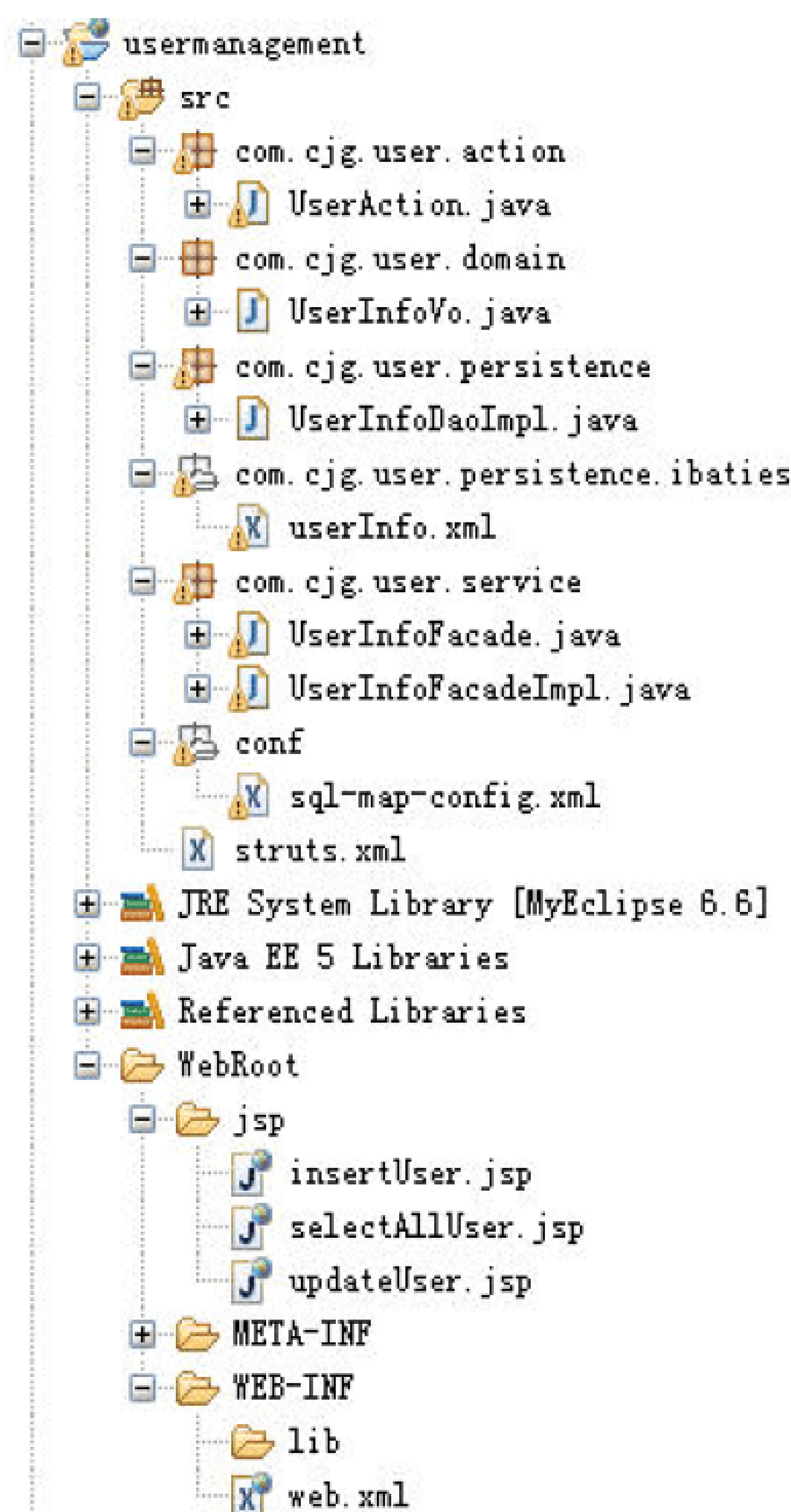


图 21.1　项目目录

21.1.2　用户维护模块功能描述

本节将以直观的方式向读者介绍整个用户维护模块要实现的功能。这些功能包括增加用户、删除用户、修改用户信息和查询所有用户信息。

1．增加用户

用户首先通过浏览 jsp 文件夹下的 insertUser.jsp 页面打开添加用户页面，如图 21.2 所示。添加相应的信息后单击“提交”按钮就可以实现用户的

增加功能。

当添加用户成功后就会直接转到图 21.3 所示的“查询所有用户”的页面。如果还想增加用户，单击“增加用户”链接就可以转到添加用户的页面。



图 21.2　用户登录界面



图 21.3　查询所有用户页面

2．修改用户信息

如果想修改用户的信息，单击“查询所有用户”页面中的“修改”链接就可以直接进入图 21.4 所示的“更新用户信息”页面。在该页面中直接修改相应信息后，单击“提交”按钮就可以直接进入如图 21.5 所示的“查询所有用户”的页面，从而实现修改用户信息功能。



图 21.4　更新用户信息页面



图 21.5　更新用户信息后页面

3．删除用户信息

如果想删除用户信息，单击“查询所有用户”页面中的“删除”链接，就可以直接进入如图 21.6 所示实现删除功能后的“查询所有用户”页面。



图 21.6　删除用户信息后页面

至此，就完成了演示用户维护的功能。

21.2　关于用户维护基础知识——iBATIS 框架

在实现数据持久化时，除了可以使用 Hibernate 框架外，还可以利用 iBATIS 框架来代替。与 Hibernate 框架相比，iBATIS 框架的最大特点就是小巧，上手容易。当需要开发的功能不复杂时，iBATIS 框架就是比较合适的解决方案。

21.2.1　下载和配置 iBATIS 框架

在具体学习 iBATIS 框架之前，首先需要下载和配置 iBATIS 类库。由于在编写该书时，iBATIS 框架的版本号为 2.3.4 版本，所以本书所有的应用都是基于 2.3.4 版本的 iBATIS 框架。

（1）首先访问下载 iBATIS 框架的官方网站（http://ibatis.apache.org/），如图 21.7 所示。

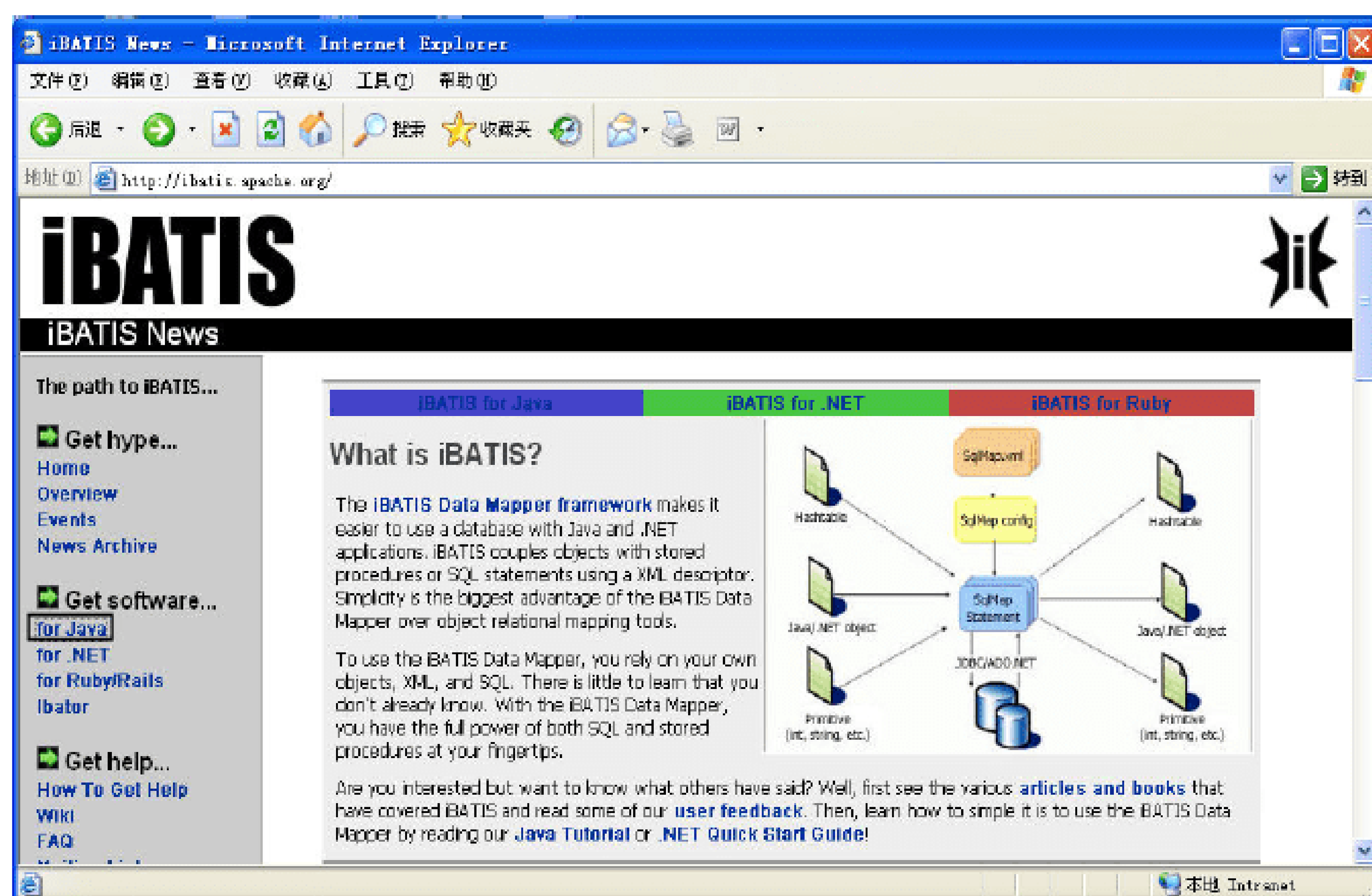


图 21.7　iBATIS 框架首页

（2）在 iBATIS 框架的官方网站首页中，单击 Get software 项目中的 for java 链接就可以进入如图 21.8 所示的关于 Java 方面的 iBATIS 框架页面。在该页面中单击 Download iBATIS Java 2.3.4 链接就可以实现该架包的下载。

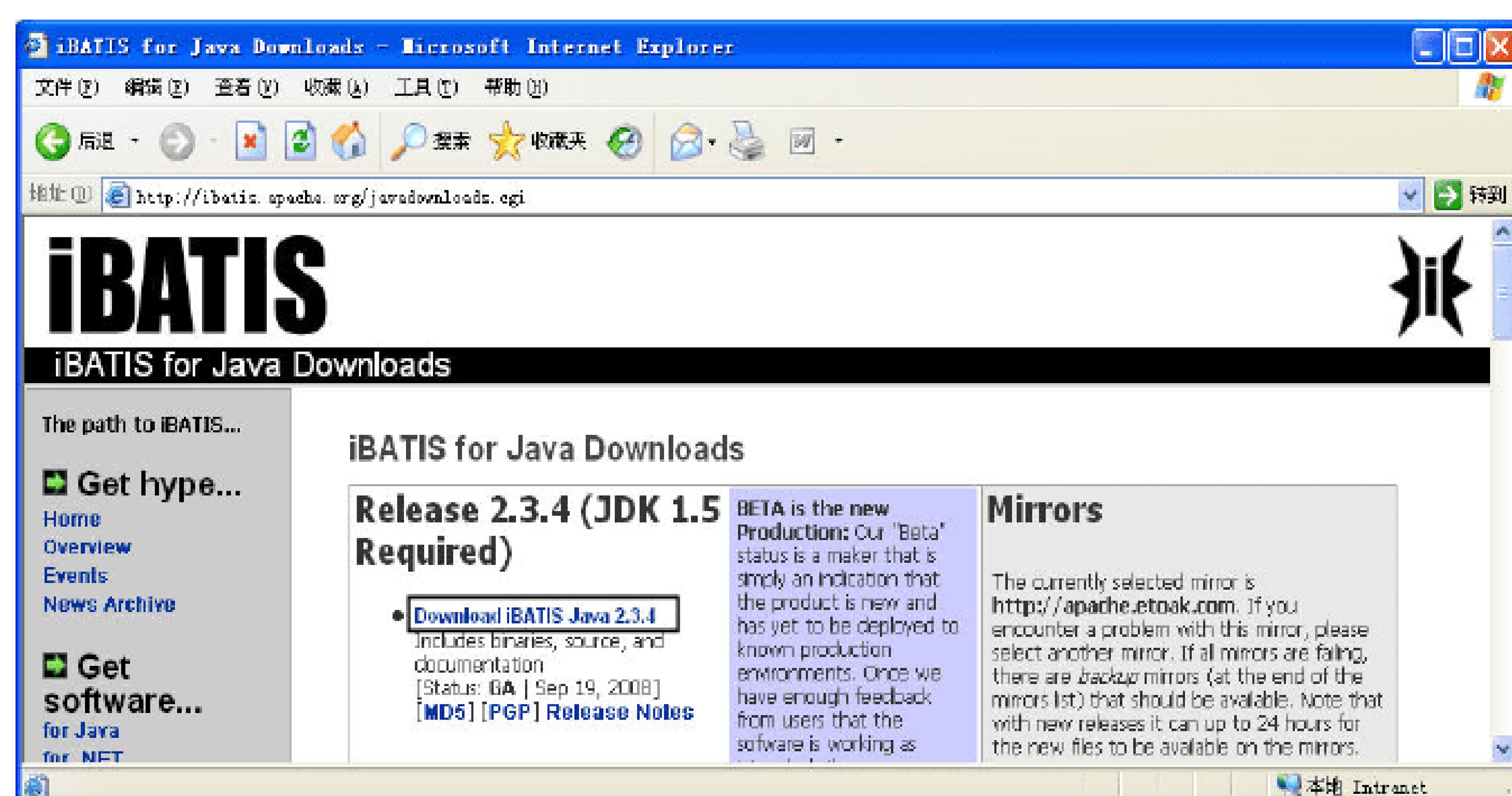


图 21.8　关于 Java 方面 iBATIS 框架页面

下载完 iBATIS 框架相关 jar 文件就可以在 Java Web 项目中使用该框架。在具体使用之前，先解压 ibatis-2.3.4.726.zip 文件，该压缩包目录如图 21.9 所示。各个文件的作用如下：

- ❑ src：关于 iBATIS 框架的 API 类文件与简介；
- ❑ simple_example：关于 iBATIS 框架的演示示例；
- ❑ lib：关于 iBATIS 框架的 jar 文件；
- ❑ doc：关于 iBATIS 框架的帮助文档。

为了使用方便，在 MyEclipse 开发环境中专门建立一个名叫 iBATIS 用户库，如图 21.10 所示。

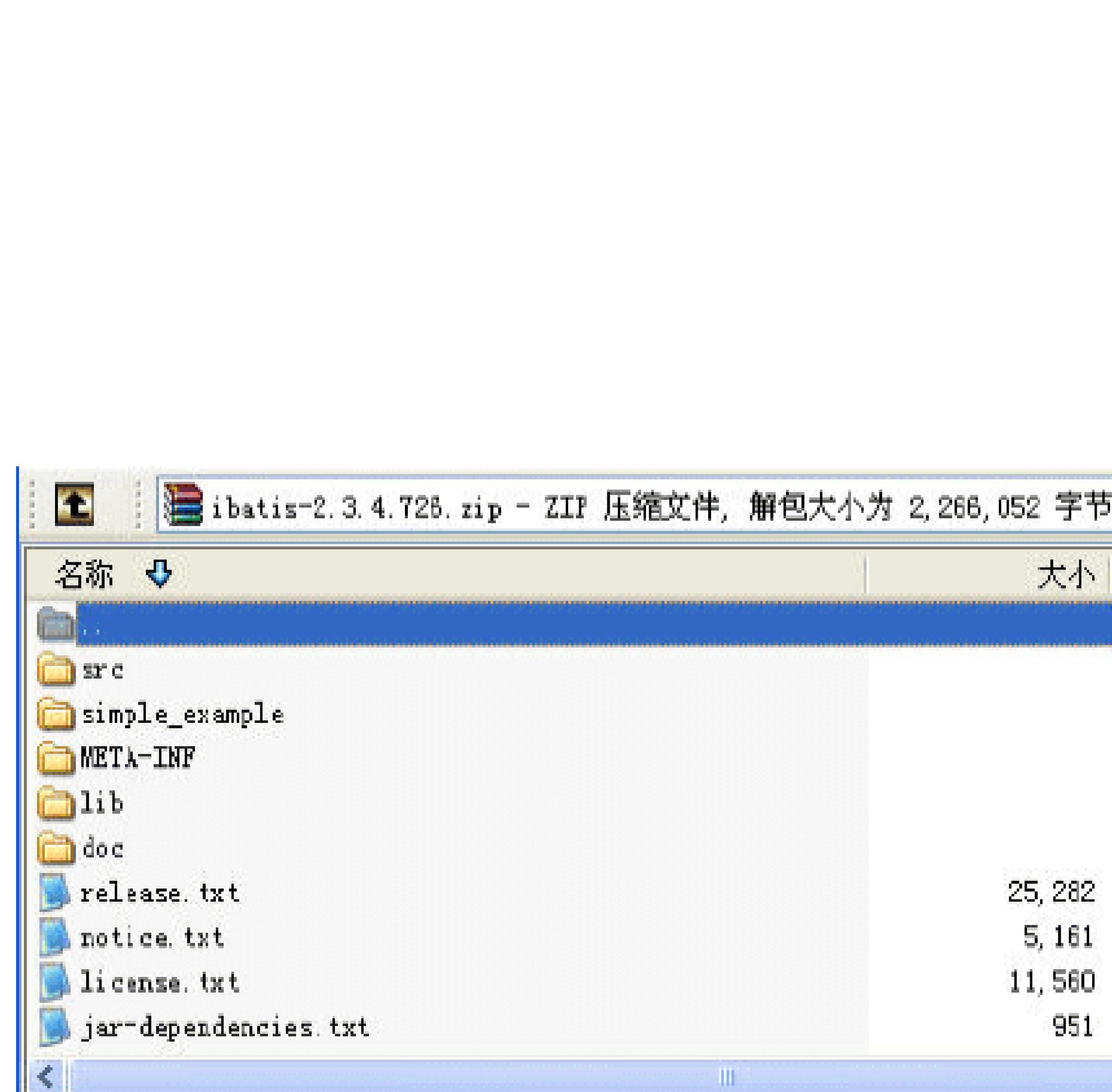


图 21.9　目录结构

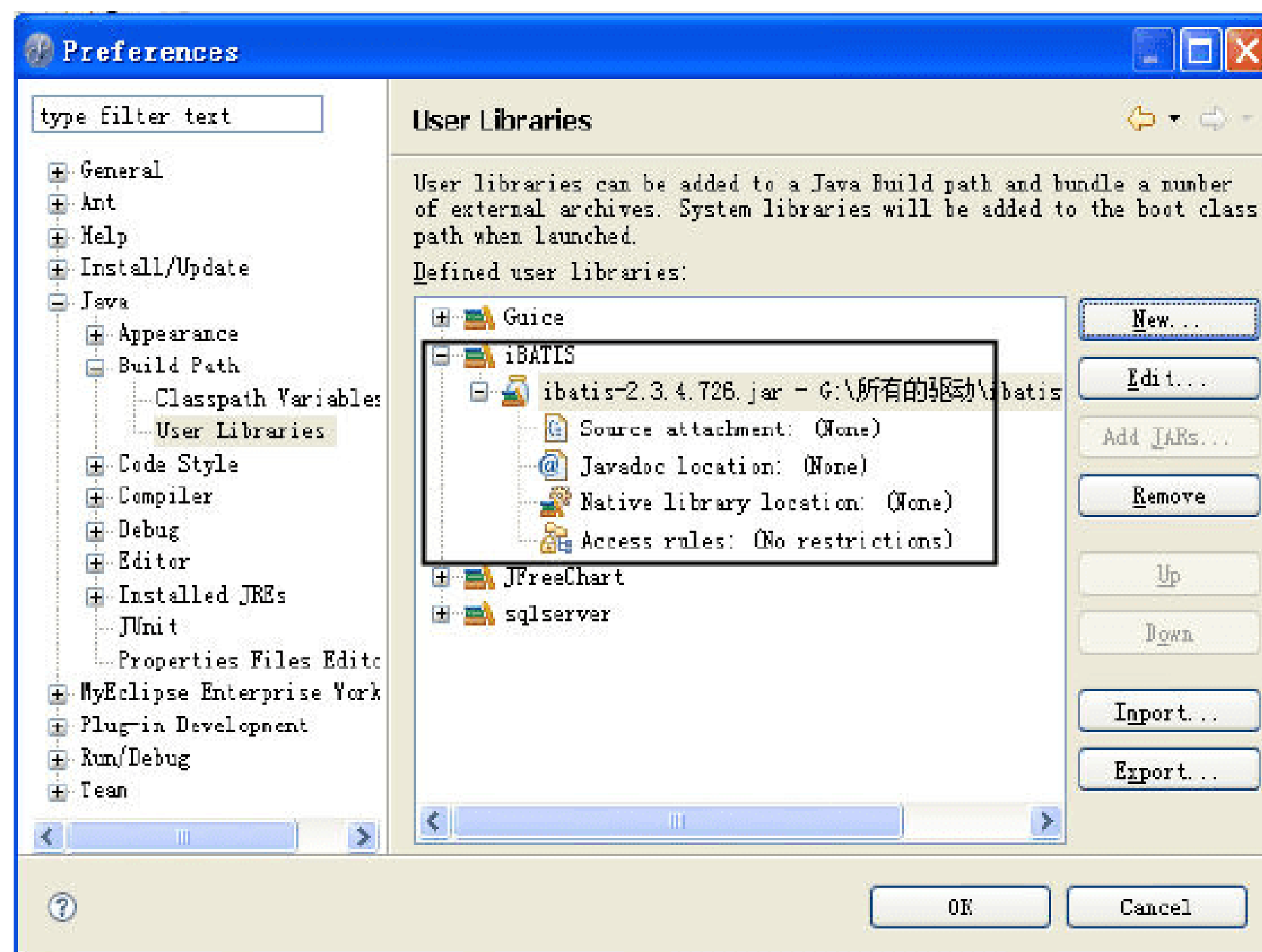


图 21.10　iBATIS 用户库

至此，就完成了下载和配置 iBATIS 相关 jar 文件。

21.2.2　iBATIS 框架的深入了解——SQL Map 数据库配置文件

在 iBATIS 框架中存在用来实现持久化的两个 XML 文件，其中一个名为 sql-map-config.xml 的 XML 文件，用来配置 iBATIS 框架自身的事务处理方式、缓冲机制等信息。由于该文件的存在，使得大大减少访问关系型数据库的代码量。该文件的具体内容如代码 21.1 所示。

代码 21.1　SQL Map 配置文件：sql-map-config.xml

```
<?xml version="1.0" encoding="GBK" standalone="no"?>
<!DOCTYPE sqlMapConfig PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN"
"http://www.ibatis.com/dtd/sql-map-config-2.dtd">

<sqlMapConfig>
<!--定义 ibatis 自身应用信息-->
  <settings
    cacheModelsEnabled="true"
    enhancementEnabled="true"
    lazyLoadingEnabled="true"
    errorTracingEnabled="true"
    maxRequests="32"
    maxSessions="10"
```

```
    maxTransactions="5"
    useStatementNamespaces="false"/>
    <!--配置 ibatis 事务与数据源-->
    <transactionManager type="JDBC">
        <dataSource type="SIMPLE">
            <!--配置数据库驱动-->
            <property name="JDBC.Driver"
                value="com.microsoft.jdbc.sqlserver.SQLServerDriver"/>
            <!--配置数据库 URL-->
            <property name="JDBC.ConnectionURL"
                value="jdbc:microsoft:sqlserver://localhost:1433;Database-
                Name=ibatis"/>
            <!--配置用户名-->
            <property name="JDBC.Username" value="sa"/>
            <!--配置密码-->
            <property name="JDBC.Password" value="root"/>
            <property name="Pool.MaximumActiveConnections"
                value="10"/>
            <property name="Pool.MaximumIdleConnections" value="5"/>
            <property name="Pool.MaximumCheckoutTime"
                value="120000"/>
            <property name="Pool.TimeToWait" value="500"/>
            <property name="Pool.PingQuery" value="select 1 from
            userinfo"/>
            <property name="Pool.PingEnabled" value="false"/>
            <property name="Pool.PingConnectionsOlderThan"
                value="1"/>
            <property name="Pool.PingConnectionsNotUsedFor"
                value="1"/>
        </dataSource>
    </transactionManager>
    <!--配置映射文件-->
    <sqlMap resource="com/cjg/user/persistence/ibaties/userInfo.xml"/>
</sqlMapConfig>
```

如果想读懂上述代码，可以查看关于 iBATIS 框架的帮助文档。查看帮助资料，实现设置 iBATIS 框架自身应用信息元素<settings>所支持的属性，如表 21.1 所示。

表 21.1　元素<settings>的属性和功能

属性名称	作　用
cacheModelsEnabled	是否启用缓存机制
enhancementEnabled	在生成 POJO 时，是否启用增强机制来增强 getter/setter 的调用效率
errorTracingEnabled	是否启用错误日记
lazyLoadingEnabled	是否启用延迟加载机制
maxRequests	允许当前最大的并发请求数
maxTransactions	允许当前最大的并发事务数
maxSessions	允许当前最大的并发 SQLMapClient，即最大的 Session 数。该属性的设置必须在 maxRequests 属性和 maxTransactions 属性之间

查看帮助资料，元素<transactionManager>用来实现事务的配置管理，在该元素中用属性 Type 制定事务管理器的具体类型，该属性的值如表 21.2 所示。

表 21.2　事务管理器

事务管理名称	作　　用
JDBC	通过 JDBC 实现事务支持
JTA	通过 JTA 服务实现全局事务管理
EXTERNAL	外部事务管理器

元素<transactionManager>中存在<dataSource>元素，该元素为数据源设置一系列参数。与元素<transactionManager>相似，<dataSource>元素中也存在属性 type，该属性指定了 dataSource 的连接类型。该属性的值如表 21.3 所示。

表 21.3　Type属性

名　　称	描　　述
SIMPLE	通过简单数据库连接池机制实现数据源
DBCP	通过 Apache DBCP 连接池组件实现数据源
JNDI	根据 JNDI Name 从容器中获取数据源

当 Type 的值为 SIMPLE 时，其他属性的配置如表 21.4 所示。

表 21.4　关于SIMPLE值配置

名　　称	描　　述
<property name="Pool.MaximumCheckoutTime" value="120000"/>	当某个任务与连接池连接时，所允许占用的最大时间。如果超过该时间，连接池将被收回
<property name="Pool.TimeToWait" value="500"/>	当程序试图从连接池获取连接时，如果该连接池没有可用的连接时，则会让程序等待，value 值则是程序等待的最长时间
<property name="Pool.PingQuery" value="select 1 from userinfo"/>	检查数据库连接状态的语句。通常情况下 value 值为一个空逻辑的语句
<property name="Pool.PingEnabled" value="false"/>	用来设置是否允许检测连接状态
<property name="Pool.PingConnectionsOlderThan" value="1"/>	用来实现对持续连接超过设定值的连接进行检测
<property name= "Pool.PingConnectionsNotUsedFor" value="1"/>	用来实现对空闲超过设定值进行检测

当 Type 的值为 DBCP 时，其他属性的配置如表 21.5 所示。

表 21.5　关于DBCP值配置

名　　称	描　　述
<property name="Pool.MaximumWait" value="12000"/>	用来设置当连接池中没有可用连接时，允许程序等待的最长时间
<property name=" Pool.ValidationQuery " value=" select 1 from userinfo "/>	用来检测连接池是否可用

续表

名　　称	描　　述
<property name="Pool.LogQbandoned" value="false"/>	用来设置连接被废弃时是否打印日志
<property name="Pool.RemoveAban-donedTimeout" value="12000"/>	用来设置空闲连接被废弃的最大超时时间
<property name="Pool.RemoveAban-doned" value="true"/>	用来设置当空闲连接超过最大时间后连接是否被废弃

对于 property 元素的公用信息，其他属性的配置如表 21.6 所示。

表 21.6　property配置

名　　称	描　　述
<property name="JDBC.Driver" value="com.microsoft.jdbc.sqlserver.SQLServerDriver"/>	用来配置 SQL Server 的驱动，其中 value 用来设置对应的驱动
<property name="JDBC.ConnectionURL" value="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=ibatis"/>	用来配置数据库的连接，其中 DatabaseName 用来设置数据库名称
<property name="JDBC.Username" value="sa"/>	用来设置数据库登录的用户名
<property name="JDBC.Password" value="root"/>	用来设置数据库登录的密码
<property name="Pool.MaximumActiveConnections" value="10"/>	用来设置数据库连接池中可维持的最大容量
<property name="Pool.MaximumIdleConnections" value="5"/>	用来设置连接池中可挂起的连接数

还可以通过另一种方法来配置数据库，即通过 properties 属性文件来配置数据库，然后利用 iBATIS 框架的占位符引入该属性文件。例如可以编写名为 ibatis.properties 的属性文件，其具体内容如下：

```
jdbc.url=jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=ibatis
```

接着可以在 sql-map-config.xml 配置文件中，利用如下代码来实现属性文件的调用。

```
<property name="url"><value>${ibatis.url}</value></property>
```

注意：在占位符${ibatis.url}中，其中 ibatis 为属性文件的名字。

21.2.3　iBATIS 框架的深入了解——SQL Map 关于 Java 类映射文件

在 iBATIS 框架中存在用来实现持久化的两个 XML 文件，其中一个配置文件用来将数据表映射成领域模型层对象。该文件一般用所要配置 JavaBean 类的名或者与之相近的名作为名字。该文件的具体内容如代码 21.2 所示。

代码 21.2　SQL Map 映射文件：userInfo.xml

```
<?xml version="1.0" encoding="GBK" standalone="no"?>
```

```
<!DOCTYPE sqlMap PUBLIC "-//iBATIS.com//DTD SQL Map 2.0//EN" "http://www.
ibatis.com/dtd/sql-map-2.dtd">
<!--配置缓存类型-->
<sqlMap namespace="UserInfoVo">
   <cacheModel id="nationInfo-cache" type="LRU">
      <flushInterval hours="24"/>
      <flushOnExecute statement="insertUser"/>
      <flushOnExecute statement="updateUser"/>
      <flushOnExecute statement="deleteUser"/>
      <property name="cache-size" value="100"/>
   </cacheModel>
   <!--配置数据表字段与持久层类-->
   <resultMap id="userinfo-result-list" class="com.cjg.user.domain.
   UserInfoVo">
      <result property="uiId" column="ui_id"/>
      <result property="username" column="username"/>
      <result property="password" column="password"/>
   </resultMap>
   <!--插入用户-->
   <insert id="insertUser">
      insert into userInfo(username,password) values (#username#,
      #password#)
   </insert>
   <!--更新用户-->
   <update id="updateUser">
      update userInfo
      set username=#username#,
      password=#password#
      where ui_id=#uiId#
   </update>
   <!--删除用户-->
   <delete id="deleteUser">
      delete userInfo
      where ui_id=#uiId#
   </delete>
   <!--查询所有用户-->
   <select id="selectUser" resultMap="userinfo-result-list">
      select * from userInfo
   </select>
   <!--根据用户 uiid 查找单个用户-->
   <select id="findSingleUser" resultMap="userinfo-result-list">
      select * from userInfo
      where ui_id=#uiId#
   </select>
</sqlMap>
```

如果想读懂上述代码，首先了解 SQL Map 的核心概念 Mapped Statement，通过其可以使用任意 SQL 语句。在具体编写时，可以利用 Mapped Statement 定义任意的 SQL 语句、parameterMap（输入）和 resultMap（输出）。Mapped Statement 的结构如下：

```
<statement id="statementName">
    [parameterClass="具体 Class 全路径"]
       [resultClass="具体结果集的 Class 全路径"]
       [parameterMap="输入参数的属性映射"]
       [resultMap="返回结果集的属性映射"]
       [cachModel="缓存机制"]
          </statement>
```

🔔注意：在具体编写时，“[]”中的内容可以选择使用。

查看相关资料，<statement>元素的配置如表 21.7 所示。

表 21.7　<statement>元素的配置

Statement 类型	属　　性	方　　法
Statement	id parameterClass resultClass parameterMap resultMap cachModel	insert update delete 所有查询方法
<insert>	id parameterClass parameterMap	insert update delete
<select>	id parameterClass resultClass parameterMap resultMap cachModel	insert update delete 所有查询方法
<update>	id parameterClass parameterMap	insert update delete
<delete>	id parameterClass parameterMap	insert update delete
<procedure>	id parameterClass resultClass parameterMap resultMap cachModel	insert update delete 所有查询方法

例如，可以编写如下代码实现删除用户的功能，具体内容如下：

```
<delete id="deleteUser">
   delete userInfo
   where ui id=#uiId#
</delete>
```

在上述代码中，Statement 类型值为<delete>，“#”中间的就是配置好的输入参数的属性，即所谓的内联参数。在具体编写 SQL 语句时，不能直接使用特殊字符：大括号（<）和小括号（>），而是直接把这些特殊符号放入 XML 的 CDATA 块中。

🔔注意：通过表 21.7 可以发现每种 Statement 类型元素中都存在一个 ID 属性，该 ID 属性是由 iBATIS 框架执行持久化操作的唯一标识。

接着了解 SQL Map 的另一个概念 resultMap，通过其可以配置映射。在具体编写时，可以利用 resultMap 属性控制数据从结果集中取出的方式、属性和字段。例如配置数据表字段与持久层类的具体内容如下：

```
<resultMap id="userinfo-result-list" class="com.cjg.user.domain.
UserInfoVo">
   <result property="uiId" column="ui_id"/>
   <result property="username" column="username"/>
   <result property="password" column="password"/>
</resultMap>
```

在上述代码中属性 property 对应 mapper statement 返回结果对象的 JavaBean 属性名，属性 column 对应于 ResultSet 中字段的名称。由于<resultMap>元素用来将查询结果映射成 JavaBean 属性，所以属性 class 用来设置 JavaBean 类的路径。

最后了解 SQL Map 的另一个概念 cacheModel，通过其可以配置缓存。通过配置 cacheModel 缓冲模式，iBATIS 框架将数据存放到缓存中，以达到节省系统资源、提升系统效率目的。例如配置缓存的具体内容如下：

```
<sqlMap namespace="UserInfoVo">
   <cacheModel id="nationInfo-cache" type="LRU">
      <flushInterval hours="24"/>
      <flushOnExecute statement="insertUser"/>
      <flushOnExecute statement="updateUser"/>
      <flushOnExecute statement="deleteUser"/>
      <property name="cache-size" value="100"/>
   </cacheModel>
```

在上述代码中出现了<cacheModel id="nationInfo-cache" type="LRU">，其中 type 属性的值用来配置缓存方式，iBATIS 框架提供的 type 属性值如表 21.8 所示。

表 21.8　type值

缓存类型	缓存值	意义
MEMORY	STRONG WEAK SOFT	该类型的缓存方式，对应的实现类为 com.ibatis.db.sqlmap.cache.memory.MemoruCacheController，会利用 Java 对象中的集合 HasMap 来保存当前需要的数据对象。当属性值为 STRONG 时是基于传统的 Java 对象的引用机制，属性值 WEAK 为 iBATIS 框架的默认值，而属性值 SOFT 是基于 SoftRefrence 缓存机制
LRU	LRU	该类型的缓存方式，对应的实现类为 com.ibatis.db.sqlmap.lru.LruCacheController，是“最近很少用”的缓存机制

其他子元素的作用如表 21.9 所示。

表 21.9　子元素

子元素	作用
<flushInterval>	该元素用来配置缓存的刷新间隔
<flushOnExecute>	该元素用来配置对应的哪些语句采用缓存机制

21.3　用户维护系统具体实现

本章通过 Struts 2.x+iBATIS 框架技术来实现用户维护系统，通过该系统可以实现添加用户、删除用户和更新用户信息等功能。在该系统中，数据库使用的是 SQL Server 2000，而 Struts 2.x 框架的版本为 Struts 2.0。

21.3.1　设计数据库

用户维护系统需要建立一个数据库并在该数据库中建立一张表，存放表的数据库 ibatis、存放用户信息的表 userinfo。

1．创建数据库ibatis

如果想创建出数据库 ibatis，可以在 SQL 2000 的查询分析器窗口输入如下命令：

```
CREATE DATABASE 'ibatis'
```

2．创建表userinfo

如果想创建出表 userinfo，可以在 SQL 2000 的查询分析器窗口中输入相关命令。该表的具体信息如表 21.10 所示。

表 21.10　表useinfo信息

字 段 名 称	数 据 类 型	字 段 说 明
ui_id	int	编号
username	varchar(50)	用户名
password	varchar(50)	密码

实现用户模型的持久化类的具体内容，如代码 21.3 所示。

代码 21.3　用户模型：UserInfoVo.java

```
...
public class UserInfoVo {
    private int uiId;                                    //创建 uild 属性
    private String username;                             //创建 username 属性
private String password;                                 //创建 password 属性
//省略属性 uild、username、password 属性
...
}
```

21.3.2　创建和配置 iBATIS 映射文件

本节将通过 iBATIS 框架实现持久层，即创建和配置 XML 文件 userInfo.xml 和

sql-map-config.xml。其中映射文件是基于数据库表 userinfo 与持久化类 UserInfoVo。

首先，需要在 sql-map-config.xml 文件中配置关于 iBATIS 框架的必要信息，包括事务处理方式、数据源、数据连接池等。该文件的具体内容如代码 21.4 所示。

代码 21.4　iBATIS 框架配置文件：sql-map-config.xml

```
<?xml version="1.0" encoding="GBK" standalone="no"?>
<!DOCTYPE sqlMapConfig PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN"
"http://www.ibatis.com/dtd/sql-map-config-2.dtd">

<sqlMapConfig>
<!--定义 ibatis 自身应用信息-->
  <settings
    cacheModelsEnabled="true"
    enhancementEnabled="true"
    maxSessions="64"
    maxTransactions="8"
    maxRequests="128"/>
    <!--配置 ibatis 事务与数据源-->
    <transactionManager type="JDBC">
        <dataSource type="SIMPLE">
            <!--数据库驱动-->
            <property name="JDBC.Driver"
                value="com.microsoft.jdbc.sqlserver.SQLServerDriver"/>
            <!--数据库 URL-->
            <property name="JDBC.ConnectionURL"
                value="jdbc:microsoft:sqlserver://localhost:1433;Database
                Name=ibatis"/>
            <!--用户名-->
            <property name="JDBC.Username" value="sa"/>
            <!--密码-->
            <property name="JDBC.Password" value="root"/>
            <property name="Pool.MaximumActiveConnections"
                value="10"/>
            <property name="Pool.MaximumIdleConnections" value="5"/>
            <property name="Pool.MaximumCheckoutTime"
                value="120000"/>
            <property name="Pool.TimeToWait" value="500"/>
            <property name="Pool.PingQuery" value="select 1 from
            userinfo"/>
            <property name="Pool.PingEnabled" value="false"/>
            <property name="Pool.PingConnectionsOlderThan"
                value="1"/>
            <property name="Pool.PingConnectionsNotUsedFor"
                value="1"/>
        </dataSource>
    </transactionManager>
    <!--配置映射文件-->
    <sqlMap resource="com/cjg/user/persistence/ibaties/userInfo.xml"/>
  </sqlMapConfig>
```

接着需要对数据库表 userinfo 和持久层类 UserInfoVo 属性进行一一映射关系，由于持久层类的名称为 UserInfoVo，所以设置该映射文件的名称为 userInfo.xml。映射文件的具体内容如代码 21.5 所示。

代码 21.5　iBATIS 框架映射文件：userInfo.xml

```
<?xml version="1.0" encoding="GBK" standalone="no"?>
<!DOCTYPE sqlMap PUBLIC "-//iBATIS.com//DTD SQL Map 2.0//EN" "http://www.
ibatis.com/dtd/sql-map-2.dtd">
<!--配置缓存类型-->
<sqlMap namespace="UserInfoVo">
    <cacheModel id="nationInfo-cache" type="LRU">
        <flushInterval hours="24"/>
        <flushOnExecute statement="insertUser"/>
        <flushOnExecute statement="updateUser"/>
        <flushOnExecute statement="deleteUser"/>
        <property name="cache-size" value="100"/>
    </cacheModel>
    <!--配置数据表字段与持久层类-->
    <resultMap id="userinfo-result-list" class="com.cjg.user.domain.
    UserInfoVo">
        <result property="uiId" column="ui_id"/>
        <result property="username" column="username"/>
        <result property="password" column="password"/>
    </resultMap>
    <!--插入用户-->
    <insert id="insertUser">
        insert into userInfo(username,password) values (#username#,
        #password#)
    </insert>
    <!--更新用户-->
    <update id="updateUser">
        update userInfo
        set username=#username#,
        password=#password#
        where ui_id=#uiId#
    </update>
    <!--删除用户-->
    <delete id="deleteUser">
        delete userInfo
        where ui_id=#uiId#
    </delete>
    <!--查询所有用户-->
    <select id="selectUser" resultMap="userinfo-result-list">
        select * from userInfo
    </select>
    <!--根据用户 uiid 查找单个用户-->
    <select id="findSingleUser" resultMap="userinfo-result-list">
        select * from userInfo
        where ui_id=#uiId#
    </select>
</sqlMap>
```

21.3.3　设计用户维护系统的 DAO 层

本节将详细介绍设计用户维护系统的 DAO 层，通过引用 iBATIS 框架中的核心客户端 sqlMapClient，来实现对数据库表 userinfo 的各种操作。

首先创建一个名为 UserInfoDaoImpl.java 的文件，在该文件中通过封装 sqlMapClient 实现对数据库表 userinfo 的增、删、改、查功能，具体内容如代码 21.6 所示。

代码 21.6　操作数据库表：UserInfoDaoImpl.java

```
...
public class UserInfoDaoImpl {
    public SqlMapClient sqlMap;                        //定义 ibaits 客户端
    public SqlMapClient getSqlMapClientTemplate() { //获得 ibaits 客户端方法
        String resource = "conf/sql-map-config.xml";
        try {
            //创建 Reader 对象
            Reader reader = Resources.getResourceAsReader(resource);
            XmlSqlMapClientBuilder xmlBuilder = new XmlSqlMapClient-
            Builder();
                        sqlMap = xmlBuilder.buildSqlMap(reader);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return sqlMap;
    }
    public void insertUser(UserInfoVo user) throws SQLException {
                                                      //增加用户方法
        getSqlMapClientTemplate().insert("insertUser", user);
    }
    public List selectUser(UserInfoVo user) throws SQLException {
                                                      //查看所有用户方法
        try {
        List list=getSqlMapClientTemplate().queryForList("selectUser",
        user);
            return list;                              //返回所有对象
            } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
    public void updateUser(UserInfoVo user) throws SQLException {
                                                      //更新用户方法
        try {
            System.out.println("id" + user.getUiId() + "dfddf"
                    + user.getPassword());            //输出相应信息
            getSqlMapClientTemplate().update("updateUser", user);
                                                      //更新用户功能
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void deleteUser(UserInfoVo user) throws SQLException {
                                                      //删除用户方法
        try {
            getSqlMapClientTemplate().delete("deleteUser", user);
                                                      //实现删除
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //查找单个用户
    public UserInfoVo findSingleUser(UserInfoVo user) throws SQLException {
        try {
            return (UserInfoVo) getSqlMapClientTemplate().queryForObject(
                    "findSingleUser", user);          //返回 UserInfoVo 对象
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
        return null;
    }
}
```

接着再编写一个名为 UserInfoFacade.java 的接口文件，在该接口中创建了操作数据库表 userinfo 的各种方法，具体内容如代码 21.7 所示。

代码 21.7　操作数据库表：UserInfoFacade.java

```
...
public interface UserInfoFacade {
    //添加用户方法
    public void insertUser(UserInfoVo user) throws SQLException;
    //查找所有用户方法
    public List selectUser(UserInfoVo user) throws SQLException;
    //更新用户方法
    public void updateUser(UserInfoVo user) throws SQLException;
    //删除用户方法
    public void deleteUser(UserInfoVo user) throws SQLException;
    //查找单个用户方法
    public UserInfoVo findSingleUser(UserInfoVo user) throws SQLException;
    //查找功能类方法
    public List selectFunctionClass(UserInfoVo userInfovo) throws SQL-
    Exception;
    //查找功能方法
    public List selectFunction(UserInfoVo userInfovo) throws SQLException;
    //获取用户编号方法
    public Object indexUserInfo(UserInfoVo userInfovo) throws SQLException;
}
```

【代码解析】

- insertUser()方法用来实现插入用户功能；
- selectUser()方法用来实现查找所有用户功能；
- updateUser()方法用来实现更新用户功能；
- deleteUser()方法用来实现删除用户功能；
- findSingleUser()方法用来实现查找单个用户功能；
- selectFunctionClass()方法用来实现查找功能类功能；
- selectFunction()方法用来实现查找功能；
- indexUserInfo()方法用来实现获取用户编号功能。

最后，编写实现接口 UserInfoFacade 的实现类，在该类中通过调用 UserInfoDaoImpl 类来实现接口中的各个方法。具体内容如代码 21.8 所示。

代码 21.8　操作数据库表：UserInfoFacadeImpl.java

```
...
public class UserInfoFacadeImpl {
    UserInfoDaoImpl userInfoDao=new UserInfoDaoImpl(); //定义持久层 DAO
    public void insertUser(UserInfoVo user) throws SQLException {
                                                          //调用增加用户方法
        userInfoDao.insertUser(user);
    }
```

```
    public List selectUser(UserInfoVo user) throws SQLException {
                                                        //调用查看用户方法
        return userInfoDao.selectUser(user);
    }
    public void updateUser(UserInfoVo user) throws SQLException {
                                                        //调用更新用户方法
      userInfoDao.updateUser(user);
    }
    public void deleteUser(UserInfoVo user) throws SQLException {
                                                        //调用删除用户方法
      userInfoDao.deleteUser(user);
    }
    //调用查找单个用户方法
    public UserInfoVo findSingleUser(UserInfoVo user) throws SQLException {
        return userInfoDao.findSingleUser(user) ;
    }
}
```

注意：在上述代码中，分别实现了 UserInfoFacade 接口中的所有方法。

21.3.4　实现页面跳转的 Action 类

本节将通过 Struts 2.0 框架来实现页面的跳转，所有的请求都会被 struts.xml 文件转发到一个名为 UserAction 的 Action 类。UserAction 类通过调用 DAO 层中的相关方法来实现业务逻辑的同时根据返回的字符串实现页面的跳转。

首先编写 UserAction.java 文件，该文件负责处理增加用户、修改用户、查询用户及删除用户功能的请求，具体内容如代码 21.9 所示。

代码 21.9　处理请求：UserAction.java

```
...
public class UserAction {
    private int uiId;                           //定义 uiId 属性
    private String username;                    //定义 username 属性
    private String password;                    //定义 password 属性
    private List list;                          //定义 list 属性
    private UserInfoVo userinfo;                //定义 userinfo 属性
    public UserAction() {                       //构造函数
        userinfo=new UserInfoVo();
    }
    // 实例化业务层对象
    UserInfoFacadeImpl userInfoFacadeImpl = new UserInfoFacadeImpl();
    //配置属性 uiId、username、password、list、userinfo
    ...
    public String insertUser() throws SQLException,    //增加用户
            UnsupportedEncodingException {
        UserInfoVo user = new UserInfoVo();        //创建 UserInfoVo 对象
        user.setUsername(username);                //设置用户名
        user.setPassword(password);                //设置密码
        userInfoFacadeImpl.insertUser(user);       //增加用户
        return "selectAllUser";
    }
```

```
    public String selectAllUser() throws SQLException {    //查看所有用户
        UserInfoVo user = new UserInfoVo();          //创建 UserInfoVo 对象
        List list1 = userInfoFacadeImpl.selectUser(user); //获取相应的对象
        setList(list1);
        return "selectUserjsp";
    }
    public String updateUser() throws SQLException {       //更新用户
        UserInfoVo user = new UserInfoVo();          //创建 UserInfoVo 对象
        user.setUsername(username);                  //设置用户名
        user.setPassword(password);                  //设置密码
        user.setUiId(uiId);                          //设置编号
        userInfoFacadeImpl.updateUser(user);         //更新用户
        return "selectAllUser";
    }
    public String deleteUser() throws SQLException {   //删除用户
        UserInfoVo user = new UserInfoVo();          //创建 UserInfoVo 对象
        user.setUiId(uiId);                          //设置用户编号
        userInfoFacadeImpl.deleteUser(user);         //删除用户
        return "selectAllUser";
    }
    public String findSingleUser() throws SQLException {   //查找单个用户
        userinfo.setUiId(uiId);                              //设置用户编号
        userinfo = userInfoFacadeImpl.findSingleUser(userinfo);//查找用户
        return "findSingleUser";
    }
}
```

接着在 struts.xml 文件中对该文件进行配置，具体内容如代码 21.10 所示。

代码 21.10　处理请求：struts.xml

```
...
<struts>
    <!-- 创建自己的包，该包必须继承默认包-->
    <package name="default" extends="struts-default">
    <!--配置 Action-->
    <action name="*" class="com.cjg.user.action.UserAction"method="{1}">
        <result name="selectAllUser" type="chain">selectAllUser</result>
        <result name="selectUserjsp">/jsp/selectAllUser.jsp</result>
        <result name="findSingleUser">/jsp/updateUser.jsp</result>
    </action>
    </package>
    </struts>
```

【代码解析】

- 根据用户维护系统所要实现的功能，在 struts.xml 文件中需要配置 5 种请求，分别为 insertUser.action：增加用户功能；selectAllUser.action：查看所有用户功能；findSingleUser.action：查找单个用户功能；updateUser.action：更新用户功能和 deleteUser.action：删除用户功能。
- 请求 findSingleUser 转向实现查看单个用户页面、请求 selectUserjsp 转向实现查看所有用户列表页面和请求 selectAllUser 转向到请求所有用户列表的页面。

注意：由于 insertUser.action、updateUser.action 和 deleteUser.action 请求返回的页面都是查询所有用户列表页面，因此使用通配符配置<action>元素的属性 Name。

21.3.5　设计表示层的相关页面

在用户维护系统中涉及 3 个页面：用来实现查询所有用户的页面 selectAllUser.jsp，具体内容如代码 21.11 所示；用来实现插入用户的页面 insertUser.jsp，具体内容如代码 21.12 所示；用来实现更新用户信息的页面 updateUser.jsp，具体内容如代码 21.13 所示。

代码 21.11　查询用户页面：selectAllUser.jsp

```
...
<body>
<div align="center" class="STYLE1">查询所有用户<br></div>
<form action="" method="get"><table width="400" border="1" align="center">
   <tr>                                       <!--表格各种字段标题-->
     <td>用户编号</td>
     <td>用户名称</td>
     <td>用户密码</td>
     <td> 操作</td>
   </tr>
   <s:iterator value="list" >                 <!--遍历变量 value-->
   <tr>
...
     <s:property value="uiId" />              <!--显示用户 ID-->
     <s:property value="username"/>           <!--显示用户名字-->
     <s:property value="password"/>           <!--显示用户密码-->
     <!--修改超级链接-->
     <a href='<s:url action="findSingleUser"><s:param name="uiId" value=
     "uiId" /></s:url>'>修改</a>
     <!--删除超级链接-->
     <a href='<s:url action="deleteUser"><s:param name="uiId" value=
     "uiId" /></s:url>'>删除</a>
    </tr>
...
   </s:iterator>
   </table>
   <!--超级链接-->
   <div align="center" class="STYLE1"><a href="jsp/insertUser.jsp" />增加
   用户</a><br></div>
</form>
</body>
...
```

代码 21.12　插入用户页面：insertUser.jsp

```
...
 <body>
 <!--表单-->
 <form name="form1" method="post"action="<%=request.getContextPath()
 %>/insertUser.action">
   <p align="center" >欢迎进入用户维护系统：增加用户</p>
...
       <td >                  <!--用户输入框-->
           <s:textfield name="username" label="用户名称"/>
       </td>
```

```
       <td >                  <!--密码输入框-->
          <s:password name="password" label="用户密码"/>
       </td>
       <td >                  <!--提交按钮-->
           <s:submit value="提交" theme="simple"></s:submit>
       </td>
       <td >                  <!--重置按钮-->
           <s:reset value="重置"theme="simple"></s:reset>
       </td>
...
 </form>
 </body>
...
```

代码 21.13　更新用户信息页面：updateUser.jsp

```
...
 <body>
 <s:form action="updateUser.action">
   <p align="center" >欢迎进入用户维护系统：更新用户信息</p>
...
      <td >                            <!--用户编号输入框-->
          <s:textfield name="uiId" value="%{userinfo.uiId}" readonly=
          "true" label="用户编号"/>
      </td>
      <td >                            <!--用户输入框-->
          <s:textfield name="username" value="%{userinfo.username}"label=
          "用户名称"/>
      </td>
      <td >                            <!--密码框-->
          <s:textfield name="password" value="%{userinfo.password}"label=
          "用户密码"/>
      </td>
      <td >                            <!--提交按钮 -->
          <s:submit value="提交" theme="simple"></s:submit>
      </td>
      <td>                             <!--重置按钮 -->
          <s:reset value="重置"theme="simple"></s:reset>
      </td>
...
 </s:form>
 </body>
...
```

21.4　小　　结

本章主要介绍用户维护系统，本系统基于 Struts 2.x+iBATIS 框架构建而成。为了让读者深入理解用户维护系统，本章主要讲解了关于持久层的框架 iBATIS 的各个方面：下载和配置 iBATIS 框架、SQL Map 数据库配置文件和 SQL Map 关于 Java 类的映射文件。在具体实现用户维护系统时，不仅通过 Struts 2.x+iBATIS 框架实现，而且还严格遵守 J2EE 框架的 4 层标准结构。

第 22 章　用户登录模块（Struts 2.x+Guice+国际化）

用户登录模块对于任何网络系统来说是一个常见而不可缺少的模块，对于每一个网络系统的前台页面程序来说几乎是必须模块。虽然任何一个程序员都能快速地编写出一个简单的用户登录模块，但是一个功能完备、性能优越的用户登录模块却不是每个程序员都能实现的。

本章将通过 Struts 2.x+Guice 框架技术来实现用户登录模块，同时利用 Struts 2.x 框架中的国际化资源来实现用户登录模块语言的多元化。

22.1　用户登录概述

对于本节的用户登录模块，由于主要用于讲解 Struts 2.0 框架中的国际资源化和 Guice 框架，所以就不涉及数据库方面的操作。

22.1.1　用户登录结构框架分析

对于一个大型网上系统来说，实现一个可用的用户登录模块要考虑的情况十分复杂，例如，如何使用户登录页面更具有人性化，如何提高用户登录模块的性能等。本系统的结构框架如图 22.1 所示。其项目目录如图 22.2 所示。

22.1.2　用户登录功能描述

本节将以直观的方式来向读者介绍整个用户登录模块要实现的功能。这些功能包括语言的多元化和用户登录功能。

用户首先通过浏览 login.jsp 页面打开用户登录页面，如图 22.3 所示。当单击“(G)英文”链接就可以转成英文用户登录界面，如图 22.4 所示。

当用户在英文用户登录界面中填写如图 22.5 所示的信息后，单击(p)submit 按钮后就可以转到如图 22.6 所示的成功页面，如果填写的信息稍有不同就会转到如图 22.7 所示的失败页面。

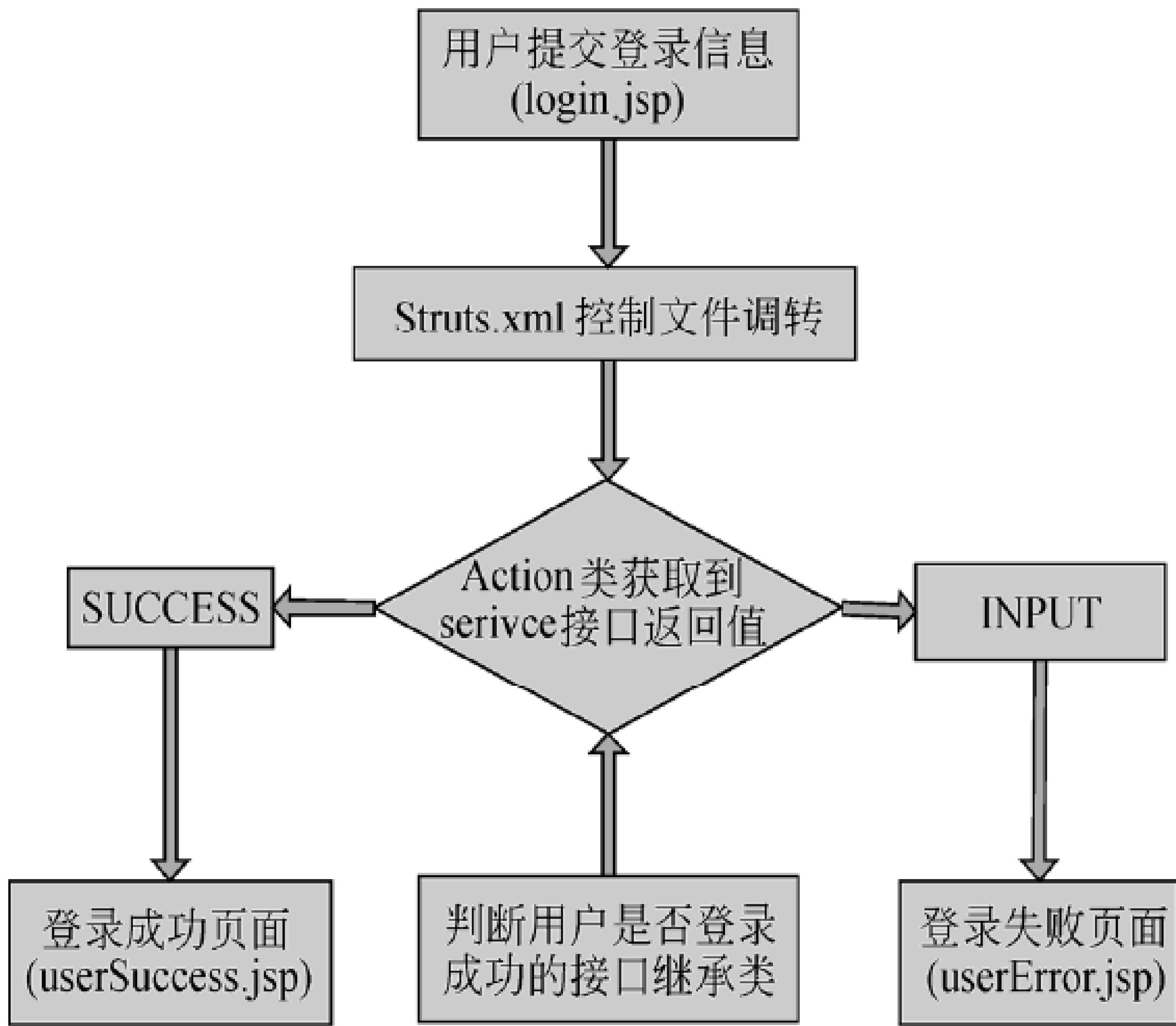


图 22.1　系统流程图

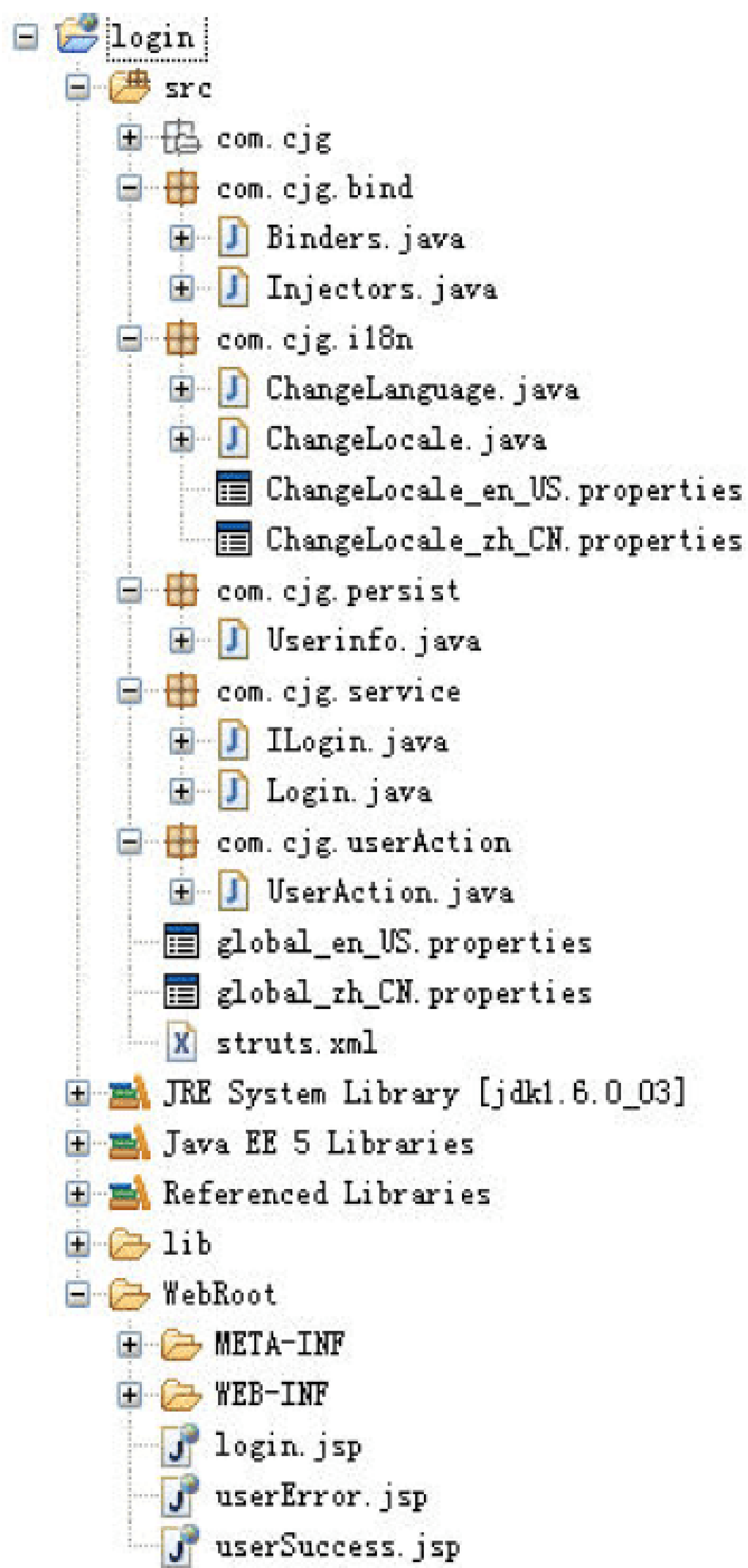


图 22.2　项目目录

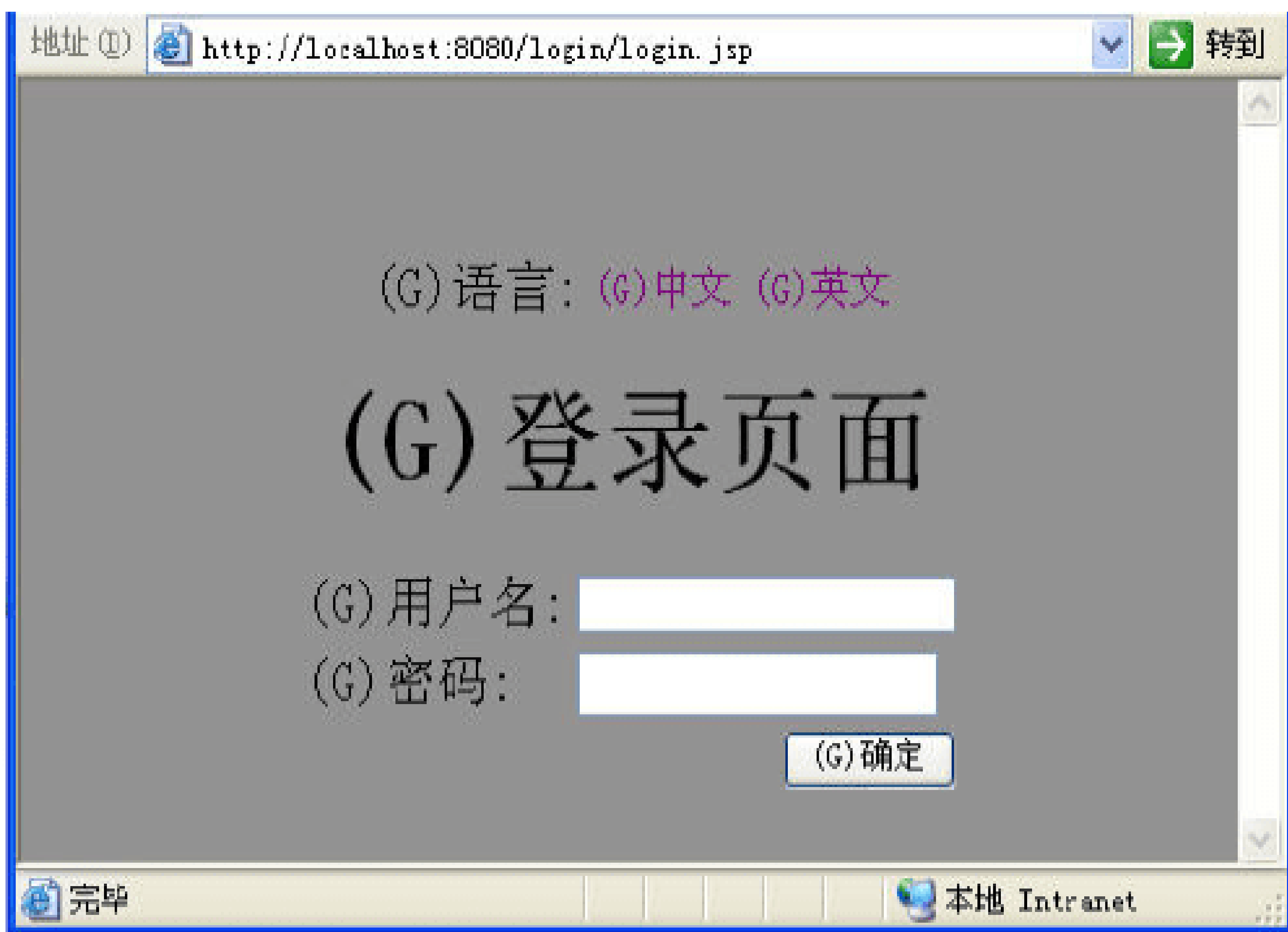


图 22.3　用户登录界面



图 22.4　英文用户登录界面

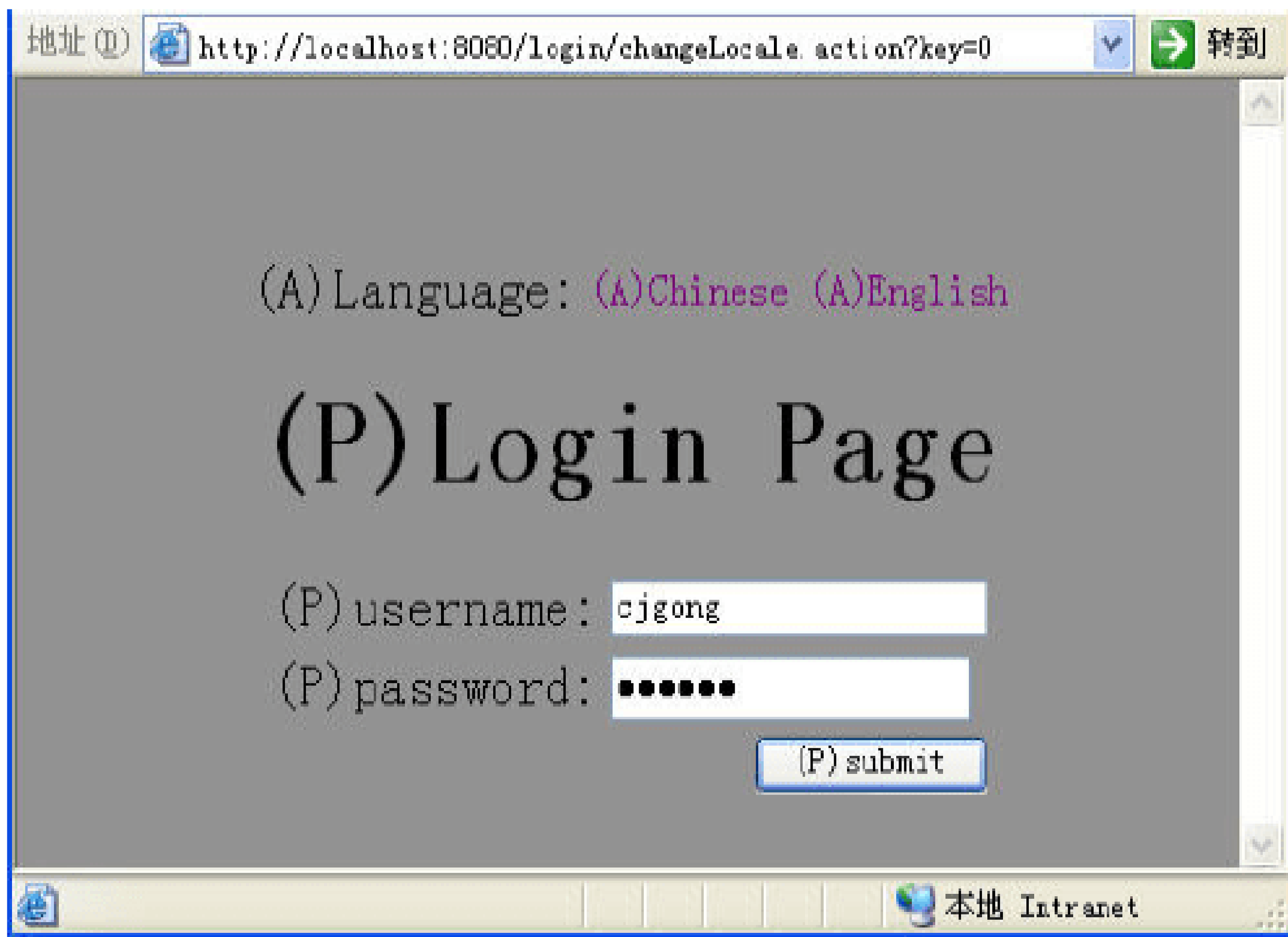


图 22.5　选择支付银行

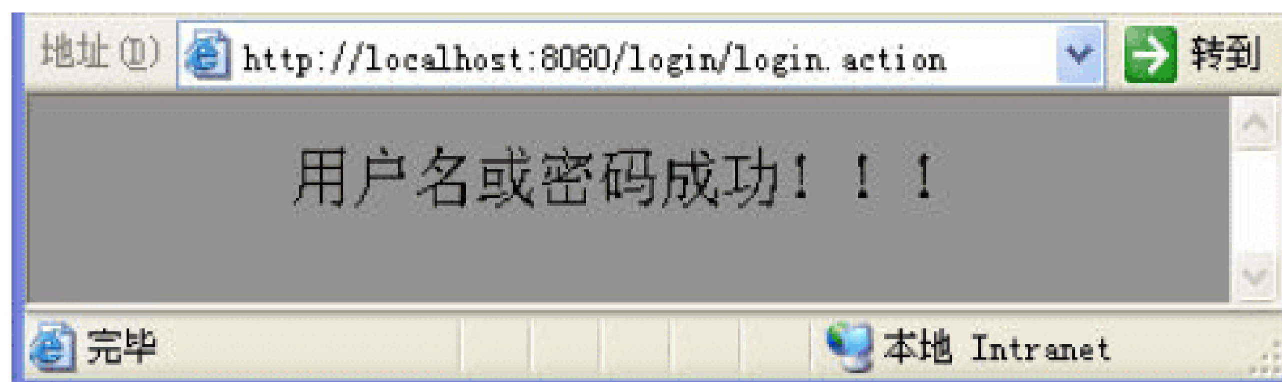


图 22.6　登录成功页面

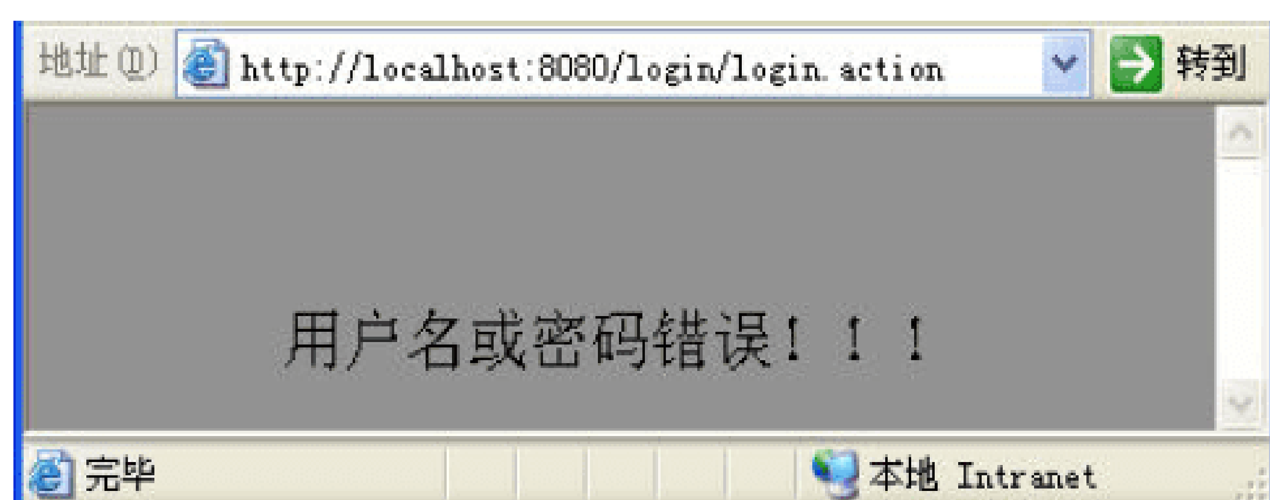


图 22.7　用户登录失败页面

22.2　关于用户登录的基础知识——国际化资源

为了实现用户登录模块语言的多元化功能，就需要利用 Struts 2.0 框架中的国际化资源来实现。本节将详细地讲解关于国际化资源的一些基础知识。

22.2.1　初步使用国际化

国际化（Internationalization，il8n）是指程序在不修改代码内部的前提下，根据不同的语言及地区显示相应的界面。国际化的提出主要是为了当向各个国家推广 Web 产品时，使来自不同国家的所有用户不受语言的影响。如果想对国际化了解清楚，必须理解如下概念。

- 国际化资源文件：该资源文件包含了对应不同区域/语言时应该显示的信息。如果给程序添加国际化，就是使该程序拥有自动选择国际化资源文件的功能；
- Locale：是 API 帮助文档中提供对应区域/语言等信息的一个类；
- ResourceBundle：是 API 帮助文档中提供加载国际化资源的一个类；
- Il8nInterceptor：该拦截器是用来负责处理 Locale 相关信息的国际化拦截器。

当实现了国际化的程序具体运行时，首先会把当前运行环境的区域/语言信息存放到 Locale 类中，然后 ResourceBundle 类根据 Locale 类中的保存信息自动搜索对应的国际化资源文件来显示。

当程序中的某个 Action 被触发时，il8n 拦截器会先于该 Action 执行，该拦截器会自动检测 Locale 信息。如果 Session 中存在 Locale 类中的信息，则将其设置为 Action 的 Locale；如果不存在，则把本机默认的 Locale 类中的信息设置为 Action 的 Locale。

下面将通过一些具体实例来讲解支持国际化的类，具体实例如下。

1. 关于Locale类

下面将通过一个具体的实例演示 Locale 类支持的国家和地区，具体内容如代码 22.1 所示。

代码 22.1　测试 Locale 类：TestLocale.java

```
import java.util.Locale;                              //引入相应包

public class TestLocale
{
    public static void main(String[] args)
```

```
    {
        Locale[] locales = Locale.getAvailableLocales();
                                                    //获取默认的 Locale 类
        //遍历默认的 Locale
        for(Locale locale : locales)
        {
            //输出相应的国家和国家语言
            System.out.println(locale.getDisplayCountry() + " : " + locale.
            getCountry()+";"+locale.getDisplayLanguage() + " : " + locale.
            getLanguage());
        }
    }
}
```

运行该代码其结果如图 22.8 所示。从运行结果中可以看到，中国的代码为 CN；中文的代码为 zh；而美国的代码为 US；英文的代码为 en。

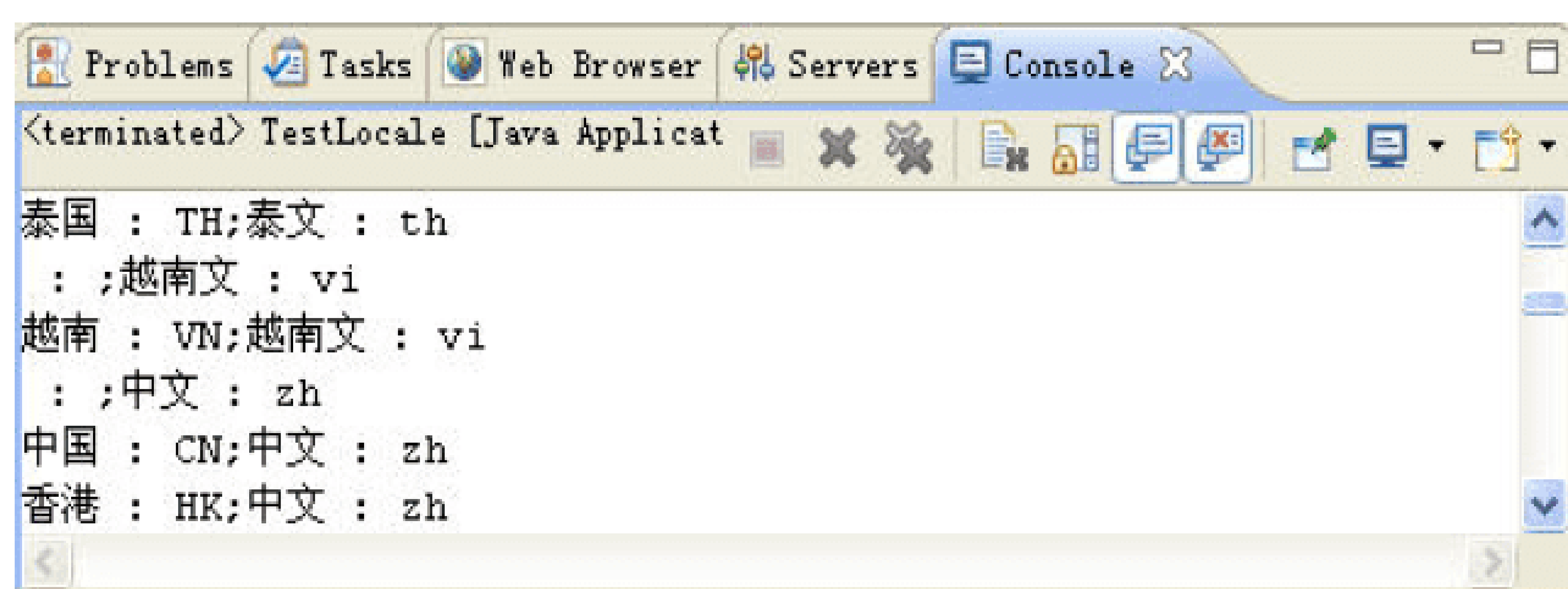


图 22.8　运行结果

【代码解析】

- 查看 JDK 的帮助文档可以发现，类 Locale 表示各个国家或地区的信息。如果想获取 Locale 类，可以通过 getAvailableLocales()方法来实现，该方法的定义如下：

```
public static Locale[] getAvailableLocales()
```

上述方法返回 Locale 实例的数组，这些实例为所有已安装语言环境的数组。

说明：虽然 Java 的类 Locale 支持绝大多数国家和地区，但是极少数的个别国家却不支持。

- 根据 Locale 类的构造函数可以知道，每个 Locale 对象由 Country（国家）和 Language（语言）组成。可以通过 getXXX()等相应的方法获取各个对象，例如 getLanguage()方法可以获取相应的语言。在 Locale 类中还存在 getDisplayXXX()等方法，这些方法是以适合用户阅读的语言显示相应的国家或语言，例如 getDisplayCountry()在默认语言环境为 zh_CN 的情况下，返回的国家为中国。

2. 关于资源文件

资源文件实际上就是符合 Java 规则的属性文件，只不过是在命名上必须符合资源文件的规范，具体规范如下。

（1）命名方式为“基本名称_语言代码_国家代码.properties”，其中基本名称是必需的，而其他两个部分却是可选的。当为同一个应用配置不同语言的资源文件时，基本名称是一致的，而语言代码和国家代码却与文件内容相一致。

（2）由于资源文件是属性文件，所以资源文件的内容结构为 key=value。在具体编写内容时，key 可以随便命名，value 则应该是同一信息不同的语言表示。

注意：对于同一应用的不同语言的资源文件内容，key 值都是相同的，但是 value 的值却是同一信息的不同语言形式。

（3）当 value 的值为非西欧字符时，必须把这些字符转换成 Unicode 编码形式。例如如果想把汉字“早上好”字符串进行转换，可以在命令行窗口通过如图 22.9 所示的命令来实现。

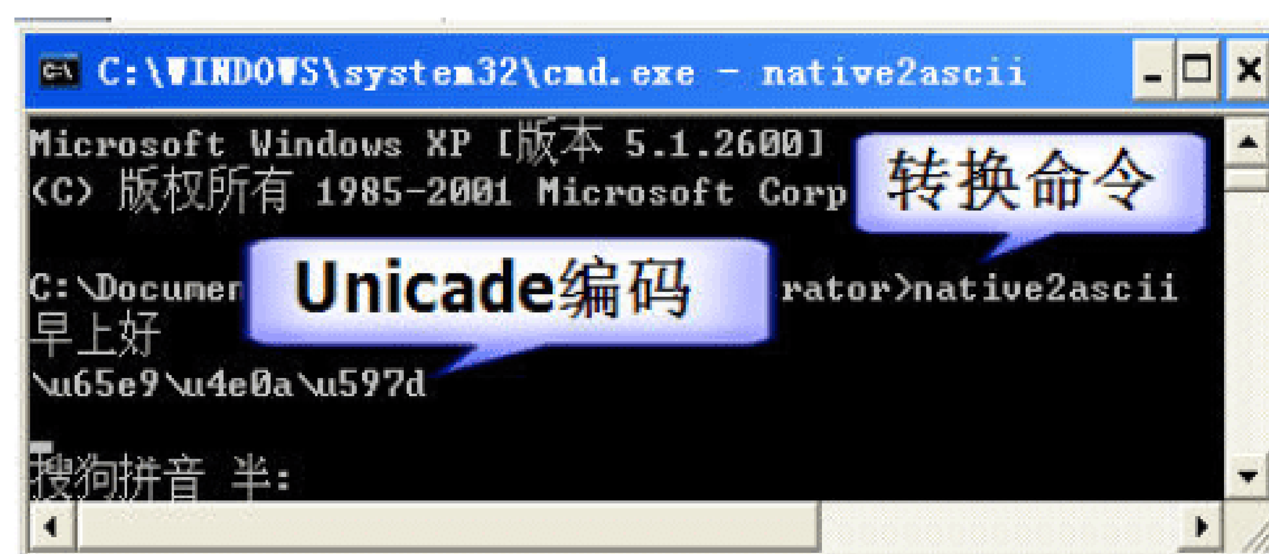


图 22.9　转换过程

当需要转换的字符串是几十条甚至上百条时，就需要使用另一种方式进行批量转换。例如如果要对 D:\a.properties 文件进行批量转换，可以在命令行窗口通过如图 22.10 所示的命令来实现。目标文件如图 22.11 所示。生成的文件如图 22.12 所示。

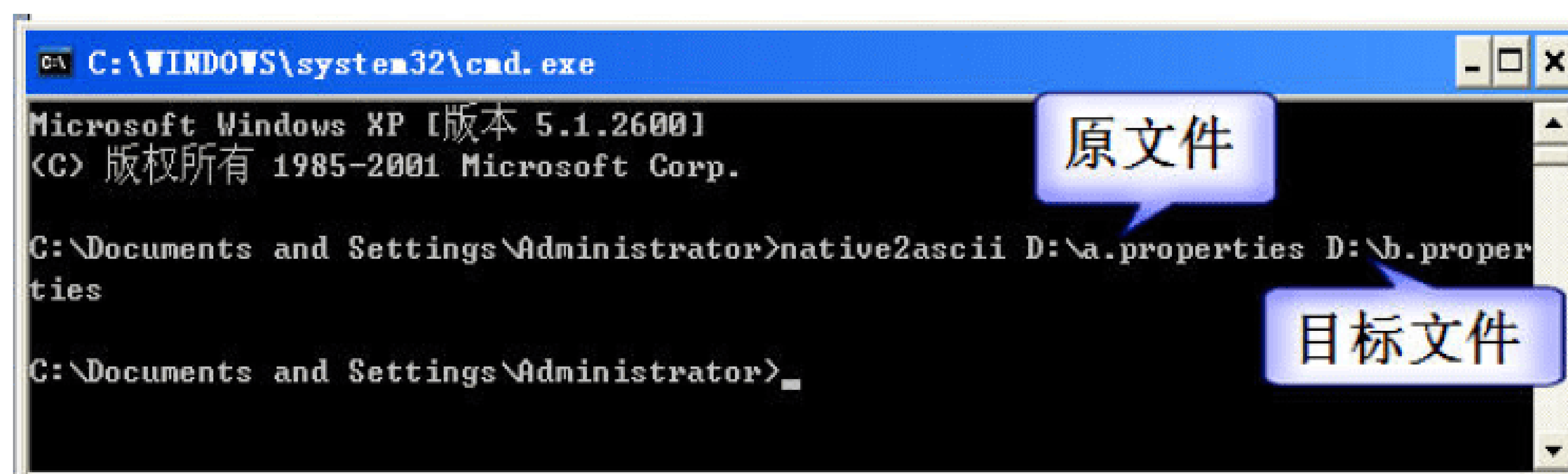


图 22.10　转换过程

通过查看 native2ascii.exe 的属性文件（如图 22.13 所示），可以发现该文件位于 JDK 安装目录下的 bin 文件夹下。如果配置了 JDK 的 path 环境变量，可以在任意路径下使用该命令，但是如果没有配置，就必须先进入 native2ascii.exe 所在的目录下，才能使用该命令。



图 22.11　生成文件



图 22.12　目标文件



图 22.13　native2ascii 属性文件

3．关于ResourceBundle类

下面将通过一个具体的实例演示利用 ResourceBundle 类来读取资源文件，具体内容如代码 22.2 所示。

代码 22.2　读取资源文件：TestResourceBundle.java

```
...
public class TestResourceBundle
{
    public static void main(String[] args)
    {
        Locale locale = Locale.getDefault();            //获取默认语言环境
        //获取资源文件
        ResourceBundle bundle = ResourceBundle.getBundle("hellofile",
        locale);
        String value = bundle.getString("hello");
                                                   //获取默认语言环境中 value 的值
        System.out.println(value);                 //输出变量 value 的值
    }
}
```

接着编写两个资源文件：hellofile_en_US.properties 和 hellofile_zh_CN.properties，这两个文件的内容分别为：

```
hello =hello world
```

和

```
hello =\u4F60\u597D
```

运行该代码其结果如图 22.14 所示。

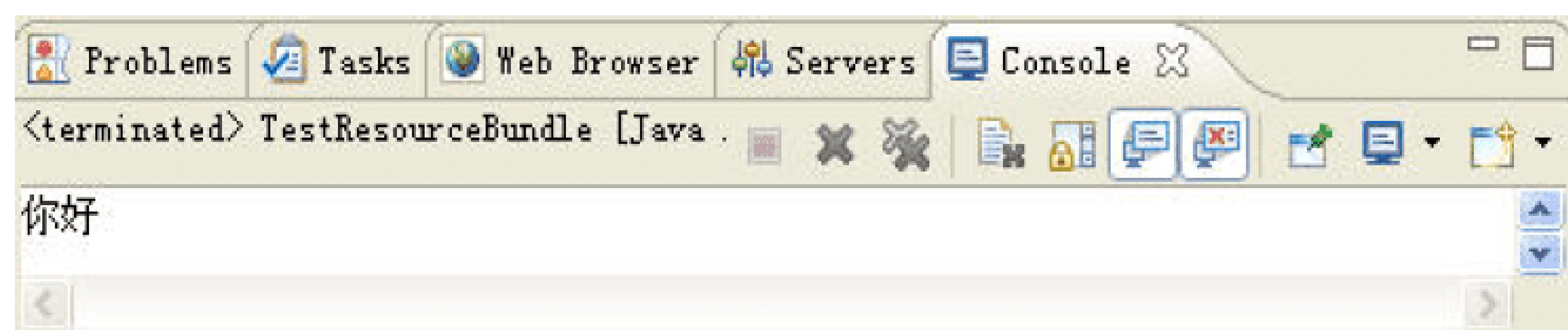


图 22.14　运行结果

【代码解析】

- 查看 JDK 的帮助文档可以发现，类 ResourceBundle 中存在一个名为 getBundle()的方法，通过该方法可以获取资源文件。该方法的定义如下：

```
public static final ResourceBundle getBundle(String baseName,
                                             Locale locale)
```

上述方法第一个参数 baseName 为资源文件名称中的基本名称，第二个参数为资源文件所需要的语言环境。

- 查看 JDK 的帮助文档可以发现，类 ResourceBundle 中存在一个名为 getString()的方法，通过该方法可以获取资源文件中特定 key 的值（value）。该方法的定义如下：

```
public final String getString(String key)
```

上述方法参数 key，为资源文件内容中所要显示 value 的 key 的名称。

- 查看 JDK 的帮助文档可以发现，通过 Locale.getDefault()可以获取当前默认的 Locale 实例。如果把该句修改成如下代码：

```
Locale locale = Locale.US;
```

则会获取美国语言环境，运行该代码则会是如图 22.15 所示的运行结果。

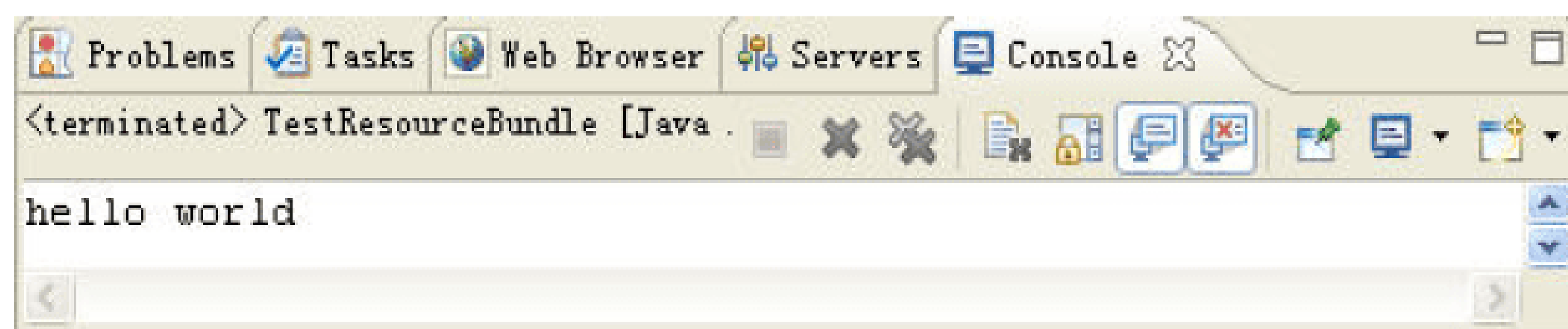


图 22.15　运行环境

在开发具体程序时，除了可以获取资源文件外，还可以传递参数到资源文件中。下面将通过一个实例来讲解如何传递参数到资源文件中，该文件的具体内容如代码 22.3 所示。

代码 22.3　读取资源文件：TestParameter

```
...
public class TestParameter
{
    public static void main(String[] args)
    {
        Locale locale = Locale.US;                     //获取美国语言环境
        //获取资源文件
        ResourceBundle bundle = ResourceBundle.getBundle("hellofile",
        locale);
        String value = bundle.getString("hello");
                                               //获取默认语言环境中 value 的值
        String result = MessageFormat.format(value,new Object[]{"北京"});
        System.out.println(result);          //输出变量 value 的值
    }
}
```

接着编写两个资源文件：hellofile_en_US.properties 和 hellofile_zh_CN.properties，这两个文件的内容分别为：

```
hello =hello world:{0}
```

和

```
hello =\u4F60\u597D:{0}
```

运行该代码其结果如图 22.16 所示。

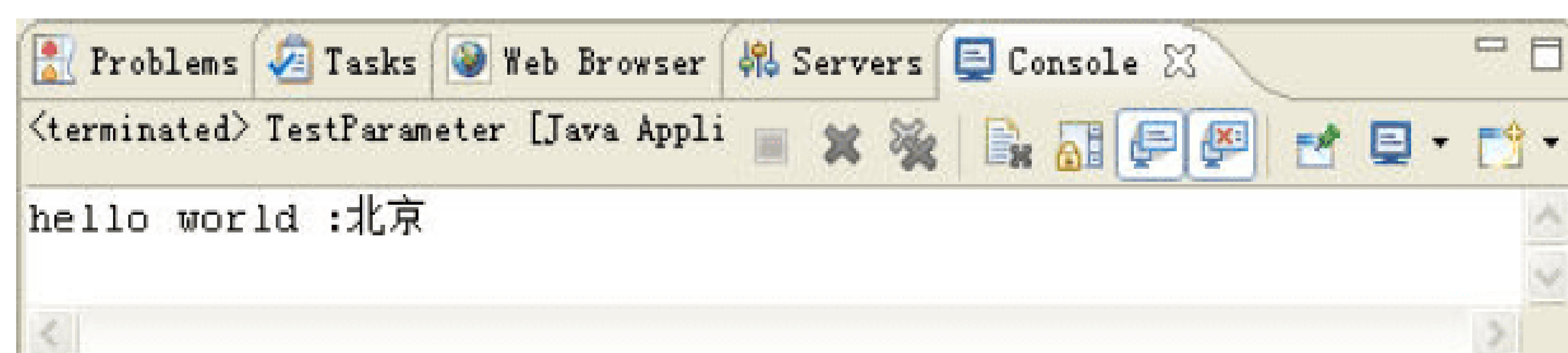


图 22.16　运行结果

【代码解析】

查看 JDK 的帮助文档可以发现，实现格式化功能的类 MessageFormat 中存在一个名为 format()的方法，通过该方法可以实现格式化参数。该方法的定义如下：

```
public static String format(String pattern,
                            Object... arguments)
```

第一个参数为字符串，第二个参数为 Object...类型（可并参数）表示 Object 的数组。具体实现时，利用第一个参数的格式来格式化第二个参数。

22.2.2　深入了解国际化——全局资源文件

在深入学习国际化中，免不了就会遇到如何调用关于国际化的资源文件、资源文件的作用范围、资源文件的分类和资源文件的调用顺序等问题。

为了解决上述问题，本节将通过一个简单的用户登录实例来深入学习国际化。具体步骤如下。

首先创建一个基于 Struts 2.x 框架的名为 il8nweb 的 Java Web 项目，在该项目中创建一个用来让用户登录的页面，具体内容如代码 22.4 所示。

代码 22.4　登录页面：login.jsp

```
...
    <body>
        <center>
            <!--超级链接-->
            <table>
                <td><s:text name="language" />:</td><td>
                <a href="changeLocale.action?key=1"><s:text name=
                "chinese" /> </a>
                <a href="changeLocale.action?key=0"><s:text name=
                "english" /> </a>
                </td>
            </table>
            <h1><s:property value="%{getText('login')}" /></h1>
                                                     <!--显示出错信息-->
            <s:actionerror/>
            <!--表单-->
            <s:form action="login">
            <s:textfield name="u.name" key="username"/>
                                                     <!--用户输入框-->
            <s:password name="u.psw" key="password"/> <!--密码输入框-->
            <s:submit key="submit"/>                 <!--提交按钮-->
            </s:form>
        </center>
    </body>
...
```

接着编写两个资源文件：global_zh_CN.properties 和 global_en_US.properties。这两个文件的内容分别为：

```
username=(G)\u7528\u6237\u540D
password=(G)\u5BC6\u7801
```

```
submit=(G)\u786E\u5B9A
chinese=(G)\u4E2D\u6587
english=(G)\u82F1\u6587
language=(G)\u8BED\u8A00
login=(G)\u767B\u5F55\u9875\u9762
firstpage=\u8FD9\u662F\u6210\u529F\u9875\u9762
```

和

```
username=(G)username
password=(G)password
submit=(G)submit
chinese=(G)Chinese
english=(G)English
language=(G)Language
login=(G)Login Page
firstpage=This is the sucess page.
```

单击工具栏上的按钮，把该项目发布到服务器中。然后单击工具栏上的按钮，启动服务器。最后打开浏览器，在地址栏中输入地址 http://localhost:8080/il8nweb/login.jsp，运行结果如图 22.17 所示。当修改本地的 Http 请求头为美国语言环境时，运行结果就会如图 22.18 所示。

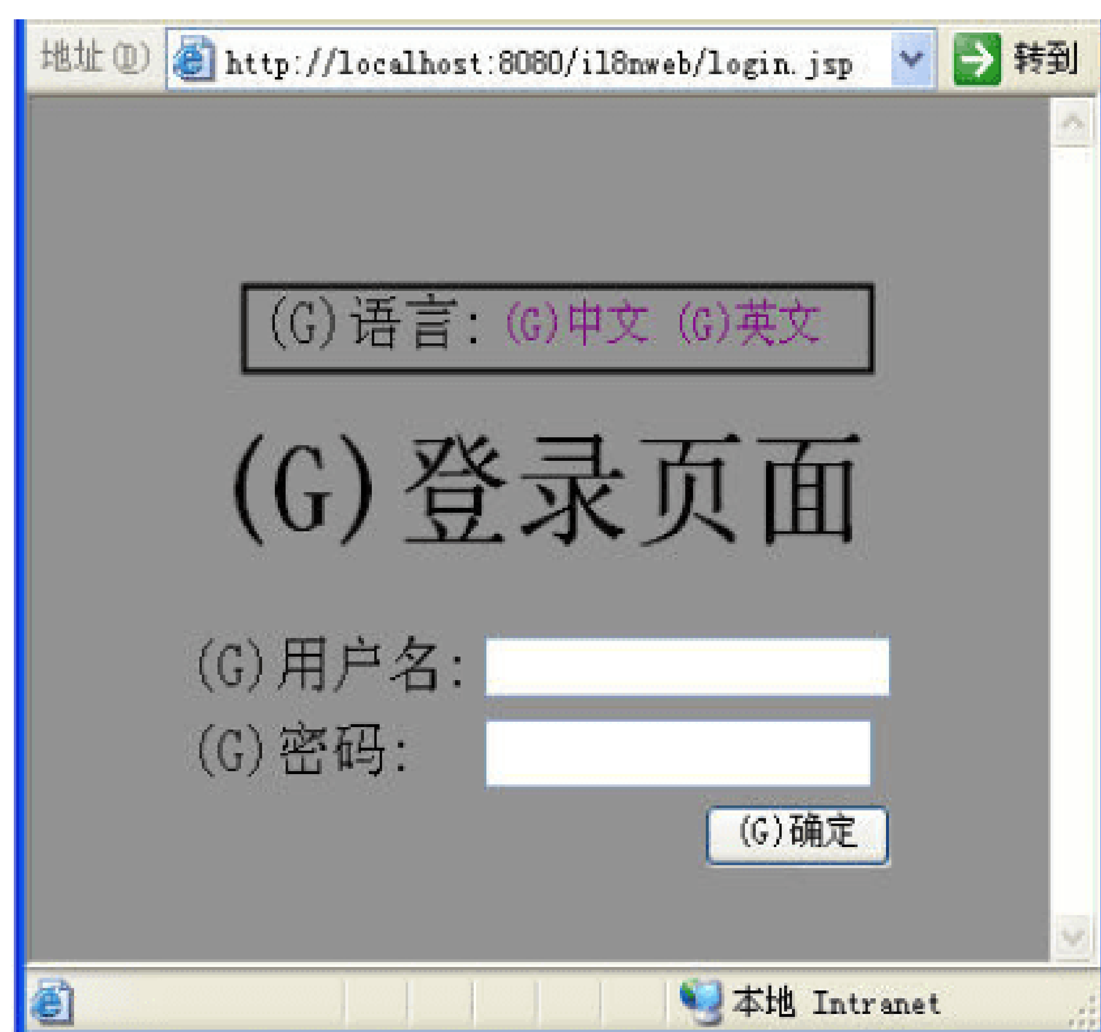


图 22.17　运行结果



图 22.18　运行结果

【代码解析】

- 查看如图 22.18 所示运行结果中的第一行，虽然代码中没有出现(G)Language、(G)Chinese、(G)English，但是在运行结果中却显示了出来。这是因为在代码中使用标签<s:text>来调用了国际化资源，如果把资源文件中的 key 值赋值给标签<s:text>的属性 name 后，当该页面被请求时，页面就会显示资源文件中相应 key 对应的 value 值；

注意：如果在资源文件中找不到 key 值，则会显示标签<s:text>中属性 name 的值。

- 查看如图 22.18 所示运行结果中的最后三行，虽然代码中没有出现(G)username、(G)password、(G)submit，但是在运行结果中却显示了出来。这是因为在 Struts 2.x 框架中的许多标签都提供了一个属性 key，如果为标签的 key 属性赋值为资源文件中的 key 值，当该页面被请求时，这些标签就会被国际化；

注意：当标签或标签所在的 form 的 themo 属性值为 simple 时，标签中 key 属性对资源文件的引用将会失败。

- 查看如图 22.18 所示运行结果的中间一行，虽然代码中没有出现(G)Login Page，但是在运行结果中却显示了出来。这是因为在代码中出现了如下代码：

```
<s:property value="%{getText('login')}" />
```

上述代码与如下代码实现的功能相同：

```
<s:text name=”login”/>
```

从上述代码中可以看出，在 JSP 页面中可以通过%{getText()}方法来获取资源文件。

注意：getText()方法不仅能够在 JSP 页面中使用，而且还能在 Action 类和校验文件中使用。

综上所述，可以通过 3 种方式来调用国际化资源文件，具体方式如下。

- 应用标签<s:text/>；
- 应用标签属性 key；
- 应用方法 getText()。

同时 global_zh_CN.properties 和 global_en_US.properties 这两个属性文件属于全局资源文件。所谓全局资源文件就是可以在整个工程范围内被使用的资源文件，该文件一般会被放在 WEB-INF/classes 路径下，同时还必须要在 struts.xml 文件中进行配置。具体配置方式如下：

```
<constant name="struts.custom.i18n.resources" value="全局资源文件的基本名称"/>
```

最后，如何修改 Http 请求头的语言环境呢？可以通过 IE 浏览器中的菜单“工具”|“Internet 选项”命令打开 Internet 选项对话框。在该对话框中（如图 22.19 所示）单击“语言”按钮就可以打开“语言首选项”对话框，在该对话框中（如图 22.20 所示）通过修改各个语言的上下位置来修改语言环境。

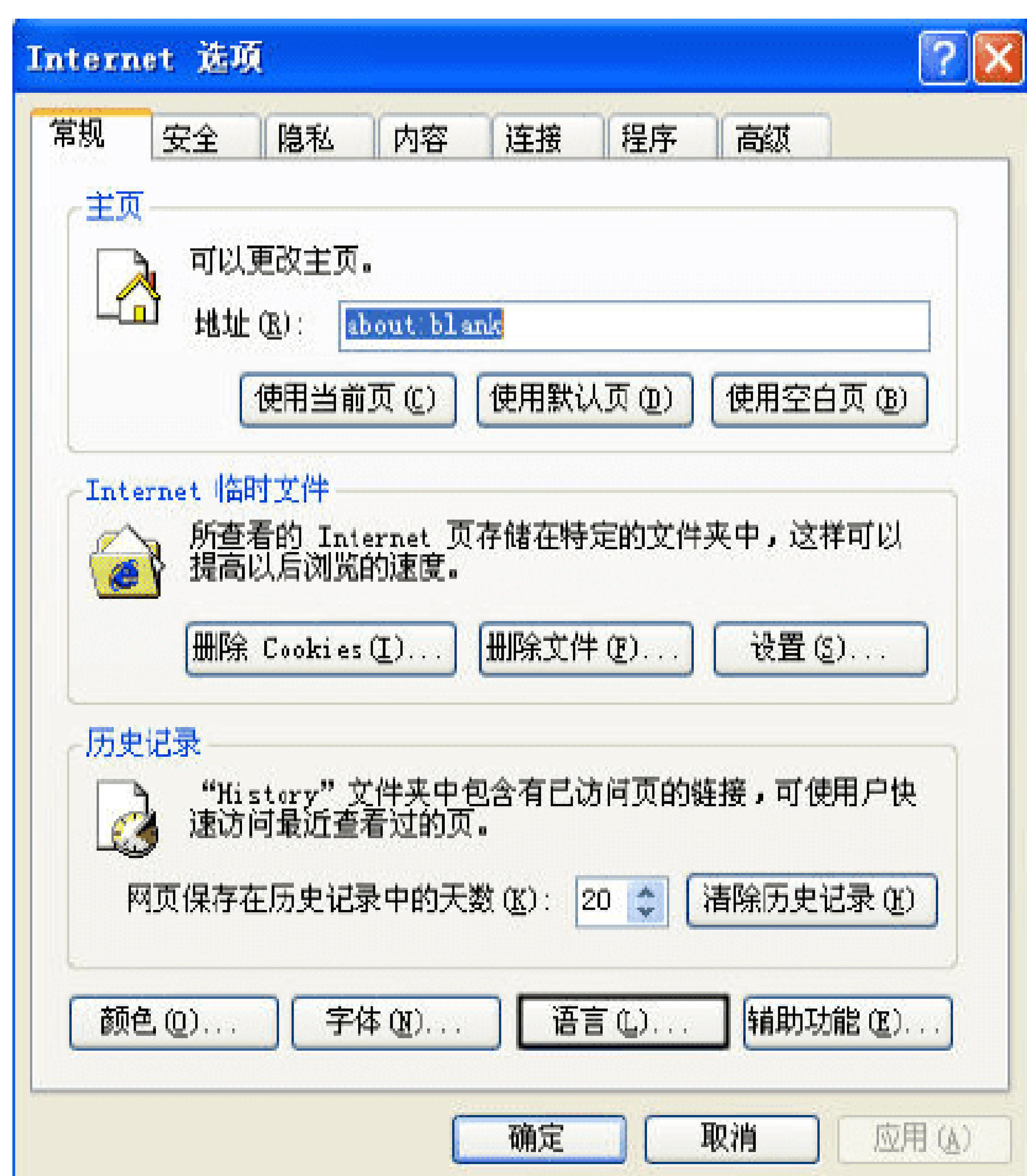


图 22.19　Internet 选项

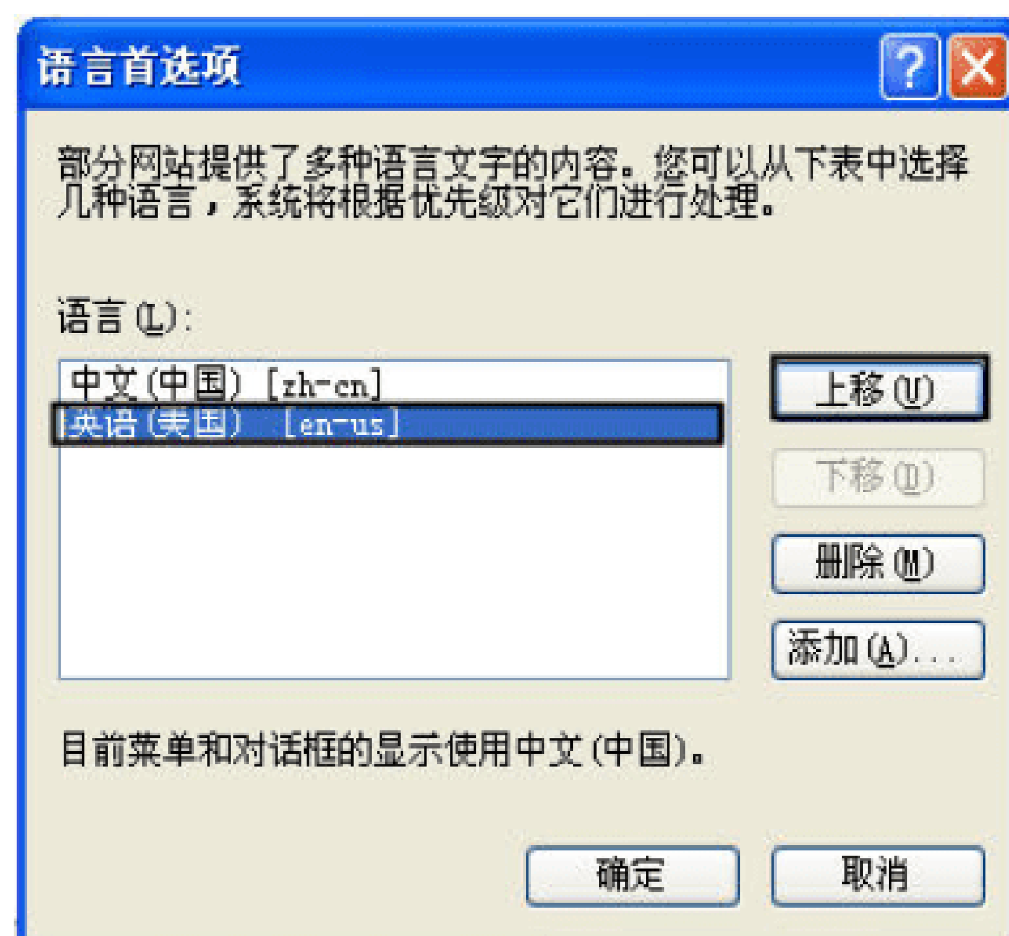


图 22.20　语言首选项

22.2.3　深入了解国际化——类资源文件

本节将接着 22.2.2 节的用户登录实例继续编写，当用户在登录页面中输入相应内容后，还需要对这些内容进行校验。具体步骤如下：

首先创建一个名为 LoginValidate 的 Action 类用来获取输入的用户名和密码，具体内容如代码 22.5 所示。

代码 22.5　获取用户名和密码：LoginValidate.java

```
...
public class LoginValidate extends ActionSupport {
    //创建相对应的属性
    private String username;
    private String password;
    //省略上述属性的 get 和 set 方法
...
}
```

接着创建校验规则配置文件，具体内容如代码 22.6 所示。

代码 22.6　校验规则配置文件：LoginValidate-validation.xml

```
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0.2//EN"
    "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

<validators>
    <!--校验用户名不为空-->
    <field name="username">
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>${getText("name.null")}</message>
        </field-validator>
    </field>
    <!--校验密码不为空-->
    <field name="password">
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>${getText("password.null")}</message>
        </field-validator>
    </field>
</validators>
```

最后编写两个资源文件：LoginValidate_zh_CN.properties 和 LoginValidate_en_US.properties，这两个文件的内容分别为：

```
name.null=(A)\u7528\u6237\u540D\u4E3A\u7A7A\uFF01
password.null=(A)\u5BC6\u7801\u4E3A\u7A7A\uFF01
```

和

```
name.null=(A)Username is empty\!
password.null=(A)Password is empty\!
```

【代码解析】

- 在具体编写校验规则配置文件时，在标签<field-validator>的子标签<message>中使用了 getText()方法来获取资源文件。在检验规则配置文件中，getText()方法的使用格式为${getText()}；
- 名为 LoginValidate_zh_CN.properties 和 LoginValidate_en_US.properties 的资源文件属于局部资源文件中类资源文件，所谓类资源文件就是指该资源文件只能被对应的类所使用。对于本节的类资源文件首先必须放在 LoginValidate.java 文件的同级目录下，而且资源文件名称中的基本名称必须为 LoginValidate。

校验完用户名和密码后，就需要对这些参数进行验证即实现登录功能，首先创建一个名为 Login 的 Action 类用来实现登录功能，具体内容如代码 22.7 所示。

代码 22.7　验证用户名和密码：Login.java

```
...
public class Login extends ActionSupport {
    private String username;                //创建 username 属性
    private String password;                //创建 password 属性
    private static final String LOGINERROR = "loginError";
                                            //创建出错字符串
    //省略上述属性的 set()和 get()方法
...
    //编写 execute()方法
    public String execute() throws Exception {
        if (username.equals("cjgong")&&password.equals("123456"))
            return SUCCESS;
        else{
            addActionError(getText("invalid"));
            return LOGINERROR;
        }
    }
}
```

接着编写两个资源文件：Login_zh_CN.properties 和 Login_en_US.properties，这两个文件的内容分别为：

```
invalid=(A)\u7528\u6237\u540D\u6216\u5BC6\u7801\u9519\u8BEF\uFF01
```

和

```
invalid=(A)Username or Password is error\!
```

【代码解析】

- 在具体编写 execute()方法时，当用户名或密码不正确时就会通过 addActionError()显示出现错误信息，该方法的参数通过资源文件来获取。在 Action 类中可以通过 getText()方法获取资源文件；
- 由于 Login_zh_CN.properties 和 Login_en_US.properties 的资源文件也属于局部资源文件中的类资源文件，所以这两个资源文件也必须要同 Login.java 文件放在同一目录下。

注意：Login.java 必须继承 ActionSupport 类，因为 getText()方法是 ActionSupport 类提供的一个方法。

综上所述，在具体使用方法 getText()调用国际化资源时，除了可以在 JSP 页面中使用外，还可以在校验文件和 Action 类中使用。

22.2.4　深入了解国际化——包资源文件

本节将接着 22.2.3 节的用户登录实例继续完善该功能。在用户登录页面中，当单击相应语言的链接后，就会调用工具类实现相应语言环境下的页面。

首先编写一个名为 ChangeLanguage 的类，实现本地语言环境与相应语言的对应，具体内容如代码 22.8 所示。

代码 22.8　实现语言对应：ChangeLanguage.java

```
...
public class ChangeLanguage extends ActionSupport {
    public Map<String, Locale> getLocales() {
        Map l = new HashMap();                      //创建 Map 对象
        l.put(getText("english"), Locale.US);
        l.put(getText("chinese"), Locale.CHINA);
        return l;
    }
}
```

接着实现相应的转换语言环境的工具类，具体内容如代码 22.9 所示。

代码 22.9　实现语言环境转变工具类：ChangeLocale.java

```
...
public class ChangeLocale extends ActionSupport {
    private String key;                             //创建属性 key
    //省略上述属性的 get()和 set()方法
    ...
    public String execute() throws Exception { //编写相应的 execute()方法
        HttpSession session = ServletActionContext.getRequest().
        getSession();
        Locale l;
        if (key.equals("1")) {                      //判断 key 的值
            l = Locale.CHINA;
        } else {
            l = Locale.US;
        }
        ActionContext.getContext().setLocale(l);    //设置语言环境
        session.setAttribute("WW_TRANS_I18N_LOCALE", l);
        return SUCCESS;
    }
}
```

接着编写两个资源文件：ChangeLocale_zh_CN.properties 和 ChangeLocale_en_US.properties，这两个文件的内容分别为：

```
chinese=(A)\u4E2D\u6587
english=(A)\u82F1\u6587
language=(A)\u8BED\u8A00
```

和

```
chinese=(A)Chinese
english=(A)English
language=(A)Language
```

【代码解析】

由于名为 ChangeLocale_zh_CN.properties 和 ChangeLocale_en_US.properties 的资源文件也属于局部资源文件中类资源文件，所以这两个资源文件也必须要同 ChangeLocale.java 文件放在同一目录下。局部资源文件除了类资源文件外，还有一种包资源文件。所谓包资源文件是指该资源文件只能被包内程序所用。对于包资源文件，首先必须放在对应包的根路基下，同时其基本名称必须为 package。

在包 com.cjg 下面编写两个资源文件：package_zh_CN.properties 和 package_en_US.properties。这两个文件的内容分别为：

```
login=(P)\u767B\u9646\u9875\u9762
username=(P)\u7528\u6237\u540D
password=(P)\u5BC6\u7801
submit=(P)\u786E\u5B9A
```

和

```
login=(P)Login Page
username=(P)username
password=(P)password
submit=(P)submit
```

上述 package_zh_CN.properties 和 package_en_US.properties 两个资源文件属于包资源文件，所以这两个资源文件的基本名称必须为 package。

22.3　关于用户登录的基础知识——Guice 框架

Google 于 2007 年 3 月发布了自己的开源项目 Guice，虽然 Guice 实现的功能与 Spring 很相似，但是该框架却在很多地方要优于 Spring 框架。本节将详细讲解一下关于 Guice 框架的基础知识。

22.3.1　下载和配置 Guice

在具体学习 Guice 框架之前，首先从下载和配置 Guice 类库开始。由于在编写该书时，Guice 框架的版本为 1.0 版本，所以本书所有的应用都是基于 1.0 版本的 Guice 框架。

（1）首先访问下载 Guice 框架的官方网站（http://code.google.com/p/google-guice/），如图 22.21 所示。

（2）在 Guice 框架的官方网站首页中，单击项目 Featured downloads 中的 guice-1.0.zip 链接就可以实现 Guice 框架相关 jar 文件的下载。

下载完 Guice 框架相关 jar 文件就可以在 Java Web 项目中使用该框架。在具体使用之前，先解压 guice-1.0.zip 文件，该压缩包目录如图 22.22 所示。各个文件的作用如下。

- javadoc：Guice 框架的帮助文档；

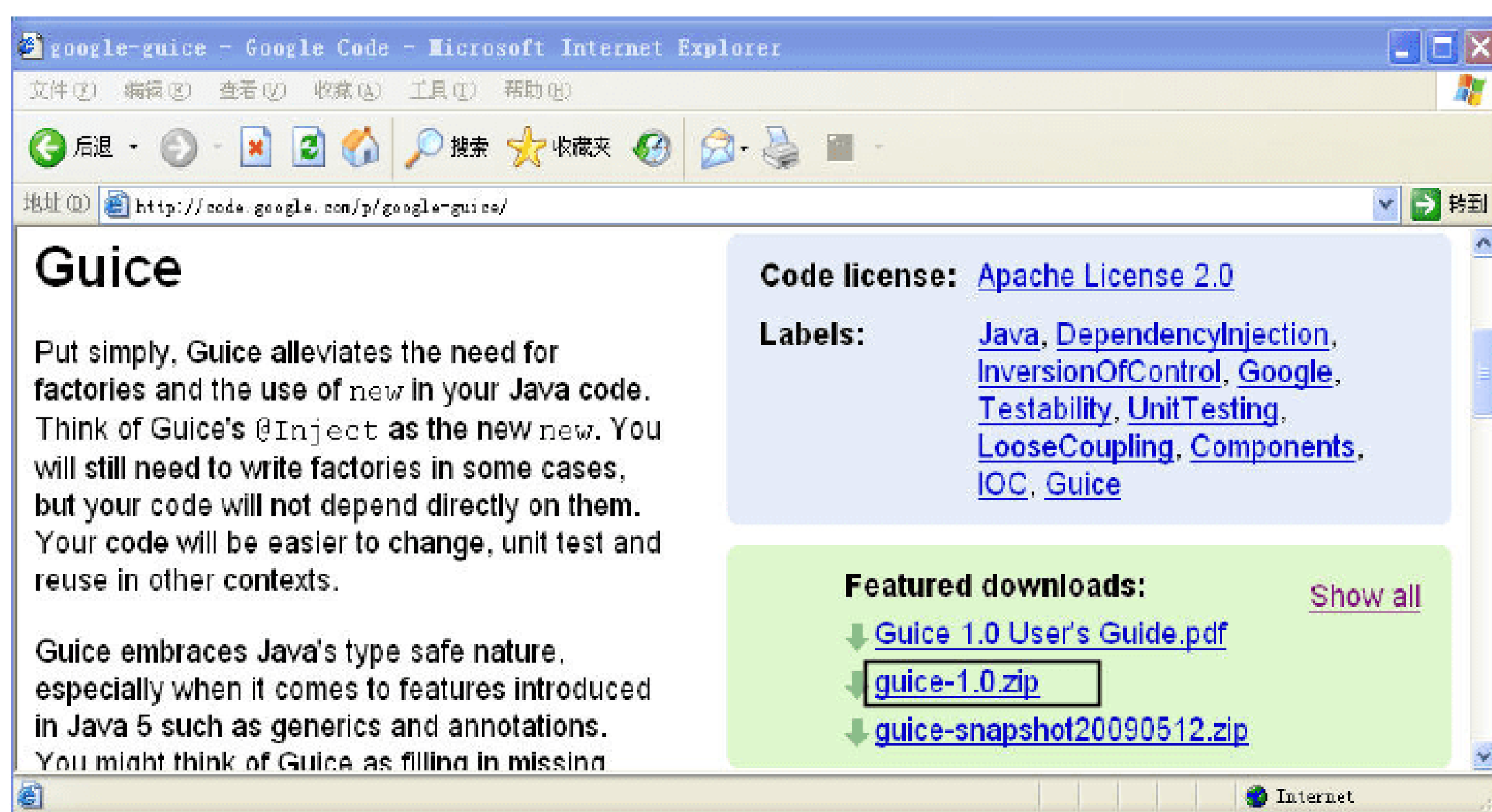


图 22.21　Guice 框架首页

- ❑ guice-struts2-plugin-1.0.jar：Guice 框架支持 Struts 2.x 的类库；
- ❑ guice-spring-1.0.jar： Guice 框架支持 Spring 的类库；
- ❑ guice-servlet-1.0.jar： Guice 框架支持 Servlert 的类库；
- ❑ guice-1.0.jar：Guice 框架的核心类库；
- ❑ aopalliance.jar：Guice 框架中关于 Aop 类库的 jar 包。

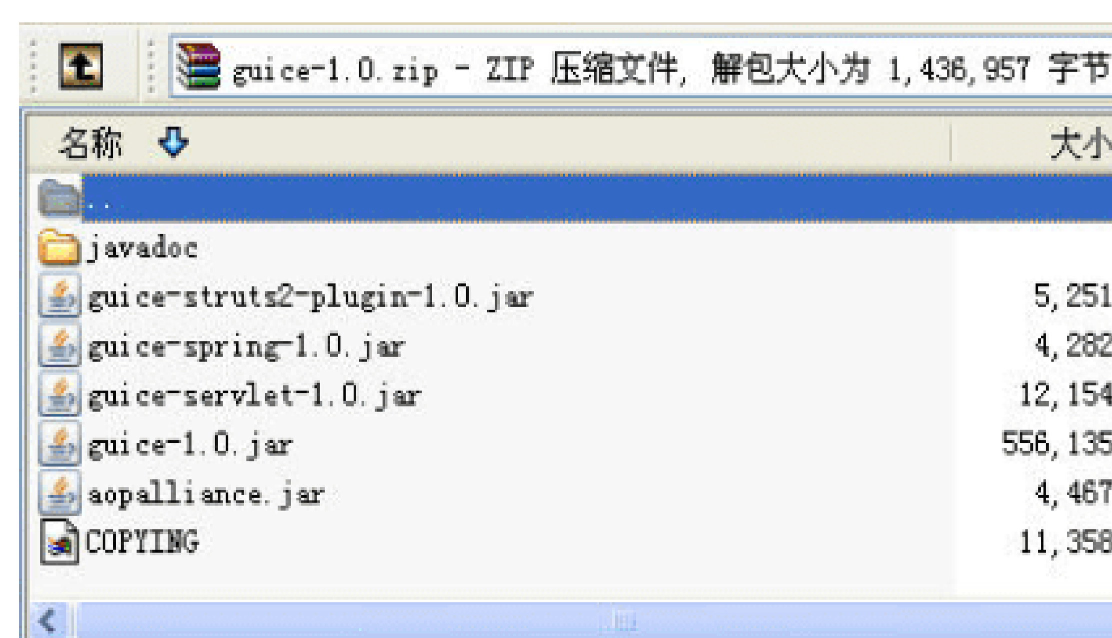


图 22.22　目录结构

为了使用方便，在 MyEclipse 开发环境中专门建立一个名叫 Guice 的用户库，如图 22.23 所示。

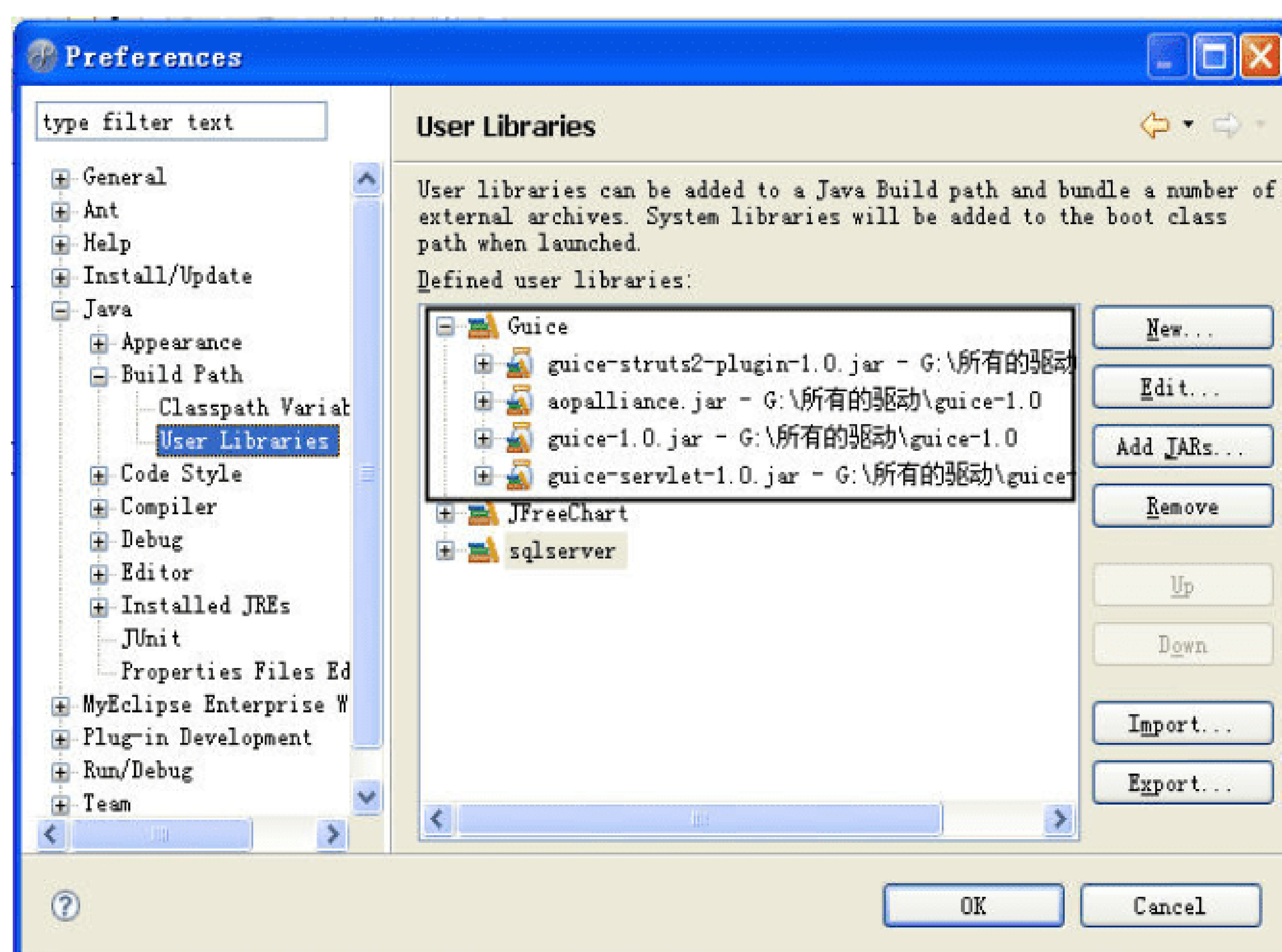


图 22.23　Guice 用户库

22.3.2　Guice 框架的简单使用

查看相关资料，可以发现 Guice 架构的使用分成两个不同的阶段：启动阶段和运行阶

段。下面将通过一个简单的输出语句实例来讲解 Guice 架构。具体步骤如下。

（1）新建一个名为 guicetest 的 Java 项目，在该项目中添加关于 Guice 框架的 jar 文件。

（2）新建一个名为 GuiceTest 的接口和继承该接口的类 GuiceTestImpl，具体内容分别如代码 22.10、代码 22.11 所示。

代码 22.10　输出语句接口：GuiceTest.java

```
public interface GuiceTest {
    public void print();
}
```

代码 22.11　实现输出语句接口：GuiceTestImpl.java

```
public class GuiceTestImpl implements GuiceTest {
    public void print() {
        System.out.println("测试 Guice 框架");
    }
}
```

（3）创建继承 Module 接口的类 Bind，具体内容如代码 22.12 所示。

代码 22.12　绑定类：Bind.java

```
//引入相应包
import com.google.inject.Binder;
import com.google.inject.Guice;
import com.google.inject.Module;

public class Bind implements Module {
    public void configure(Binder binder) {      //绑定接口和该接口继承类
        binder.bind(GuiceTest.class).to(GuiceTestImpl.class);
    }
    public static void main(String[] args) {    //实现接口和该接口继承类的注入
        Guice.createInjector(new Bind()).getInstance(GuiceTest.class).
        print();
    }
}
```

运行 Bind.java 文件，结果如图 22.24 所示。

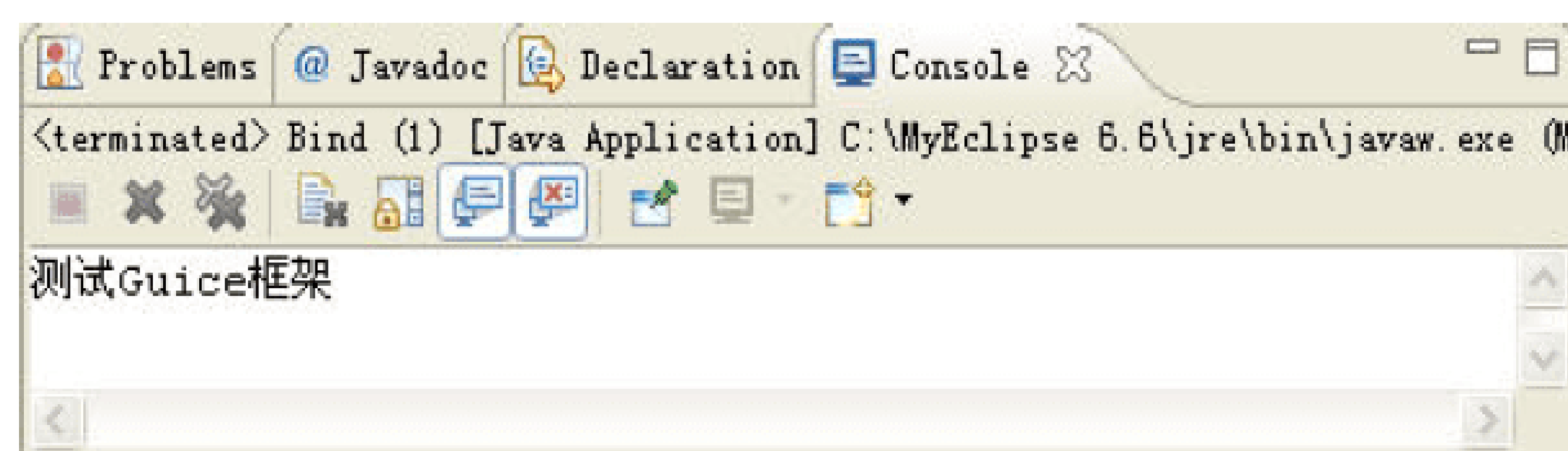


图 22.24　运行结果

【代码解析】

- 首先为了实现接口及其实现类之间的绑定，必须创建继承 Module 接口实现绑定功能的类。具体方式如下：

```
public class 绑定类 implements Module {
    public void configure(Binder binder) {
```

```
        binder.bind(接口.class).to(实现类.class);
    }
}
```

- 接着当实现接口及其实现类之间的绑定后，就需要调用 com.google.inject 包中 Guice 类的 createInjector()方法来实现简单的注入，具体方式如下：

```
Guice.createInjector(new Bind())
```

查看帮助文档，可以发现 createInjector()方法定义如下：

```
public static Injector createInjector(Iterable<Module> modules)
```

- 最后，调用包 com.google.inject 中 Injector 接口的 getInstance()方法调用绑定接口。具体方式如下：

```
Guice.createInjector(new Bind()).getInstance(接口类)
```

查看帮助文档，可以发现 getInstance()方法定义如下：

```
<T> T getInstance(Class<T> type)
```

综上所述，可以发现编写关于 Guice 的架构实例流程：首先根据业务需求定义业务接口及其实现类。然后再创建 Module 接口实现类，即覆盖 Module 接口中的 configure()方法，在该方法中通过 Binder 类进行具体绑定。接着通过 com.google.inject.createInjector()方法实现注入功能。最后就可以调用绑定接口。

注意：通常情况下，实现 Module 接口的类称为绑定类。

22.4　用户登录的具体实现

本章通过 Struts 2.x+Guice 框架技术来实现用户登录功能，其中 Struts 2.x 框架用来实现页面的跳转，而 Guice 框架则用来实现用户登录功能。

22.4.1　登录页面

用户要实现登录功能，必须要浏览登录页面，在该页面中有两个链接、两个文本框和一个提交按钮。用户可以通过链接来实现语言的多元化，同时如果要登录该系统，必须在两个文本框中输入相应的信息。登录页面的具体内容如代码 22.13 所示。

代码 22.13　实现登录页面：login.jsp

```
...
    <body>
        <center>
            <table>
                <td><s:text name="language" />:</td><td>
                <a href="changeLocale.action?key=1"><s:text name=
                "chinese" /> </a>
                <a href="changeLocale.action?key=0"><s:text name=
```

```
                "english" /> </a>
                </td>
            </table>
            <h1><s:property value="%{getText('login')}" /></h1>
                                                        <!--显示出错信息-->
            <s:actionerror/>
            <!--表单-->
            <s:form action="login">
            <s:textfield name="u.name" key="username"/>
                                                        <!--用户输入框-->
            <s:password name="u.psw" key="password"/>   <!--密码输入框-->
            <s:submit key="submit"/>                    <!--提交按钮-->
            </s:form>
        </center>
    </body>
...
```

【代码解析】

在上述代码中，对于国际化的内容就不多解释了，前面已经介绍过。当单击“登录”按钮后，u.name 和 u.psw 这两个参数将与请求 login 一起发送。那么请求 login 将由哪个 Action 类进行处理？可以查看 struts.xml 文件的具体内容。关于 login 请求的内容如代码 22.14 所示。

代码 22.14　关于请求 login 请求：struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
...
    <package name="login" extends="struts-default">
...
        <!--配置名为 login 的 Action -->
        <action name="login" class="com.cjg.userAction.UserAction">
            <result name="input">/userError.jsp</result>
            <result>/userSuccess.jsp</result>
        </action>
    </package>
</struts>
```

22.4.2　实现用户验证——处理请求过程

当 Struts 2.x 框架拦截到 login 请求后，就会由 struts.xml 文件把请求转发到名为 UserAction 的 Action 类中。在该类中会调用相应的方法来判断用户名（u.name）和密码（u.psw）是否正确。

由于验证过程是通过 Guice 框架来实现的，所以需要编写相应的接口、实现类和绑定类。而在具体编写这些类之前，还需要创建一个用来将用户的信息封装起的 JavaBean，具体内容如代码 22.15 所示。

代码 22.15　封装用户信息：Userinfo.java

```
...
public class Userinfo implements java.io.Serializable {
    private String name;                        //用户名属性
    private String psw;                         //密码属性
    //省略上述属性的 get()和 set()方法
...
}
```

具体实现登录功能的接口类和继承接口类的内容，分别如代码 22.16 和代码 22.17 所示

代码 22.16　登录功能接口：ILogin.java

```
...
public interface ILogin {
    public boolean login(Userinfo u);           //登录功能方法
}
```

代码 22.17　登录功能实现类：Login.java

```
...
public class Login implements ILogin {
    public boolean login(Userinfo u) {          //实现登录功能
        //验证用户名和密码
        if (u.getName().equals("cjgong") && u.getPsw().equals("123456")) {
            return true;
        } else
            return false;
    }
}
```

当创建完相应的接口和实现类后，就需要通过调用 Guice 框架中的相关类来创建绑定类，在该类中对接口 ILogin 和接口实现类 Login 进行了绑定，具体内容如代码 22.18 所示。

代码 22.18　绑定类：Binders.java

```
...
public class Binders extends AbstractModule {
    protected void configure() {
        bind(ILogin.class).to(Login.class);     //实现接口和接口实现类绑定
    }
}
```

当对接口 ILogin 和接口实现类 Login 进行绑定后，还需要对其进行注入。实现注入功能 Injectors 类的具体内容如代码 22.19 所示。

代码 22.19　注入类：Injectors.java

```
...
public class Injectors {
    public Injector inject() {
        return Guice.createInjector(new Binders());    //实现注入功能
```

```
    }
}
```

至此，利用 Guice 框架就完成了登录功能。

22.4.3　控制页面跳转 Action 类及其相关页面

当验证完用户名和密码后，就会返回信息给名为 UserAction 的 Action 类。接着会在 Action 类中进行判断登录是否成功，如果成功则返回 SUCCESS；否则返回 INPUT。UserAction 文件的具体内容如代码 22.20 所示。

代码 22.20　页面跳转：UserAction.java

```
...
public class UserAction {
    Userinfo u;                                    //创建 Userinfo 属性
    public Userinfo getU() {                       //配置 Userinfo 属性
        return u;
    }
    public void setU(Userinfo u) {
        this.u = u;
    }
    public String execute() {
        Injectors in = new Injectors();            //调用注入类
        if (in.inject().getInstance(ILogin.class).login(u)) {  //判断登录
            return Action.SUCCESS;                 //返回字符串 SUCCESS
        } else
            return Action.INPUT;                   //返回字符串 INPUT
    }
}
```

当返回 SUCCESS 字符串后，就会跳转到名为 userSuccess 登录成功页面；当返回 INPUT 字符串后，就会跳转到名为 userError 的登录失败页面。userSuccess.jsp 页面的具体内容如代码 22.21 所示。而 userError 页面的具体内容如代码 22.22 所示。

代码 22.21　登录成功页面：userSuccess.jsp

```
...
    <body>
        <center>
            用户名或密码成功！！！<!--显示成功信息-->
        </center>
    </body>
```

代码 22.22　登录失败页面：userError.jsp

```
...
    <body>
        <center>
            用户名或密码错误！！！<!--显示出错信息-->
        </center>
    </body>
```

至此，利用 Guice 框架结合 Struts 2.x 框架就完成了对用户登录系统的构建。

22.5　小　　结

本章主要介绍用户登录系统，该系统不同于普通的用户登录系统，其基于 Struts 2.x+Guice+国际化框架构建而成。为了让读者深入理解本章的用户登录系统，不仅讲解了关于业务层框架 Guice，而且还详细讲解了 Struts 2.x 框架中国际化：全局资源文件、包资源文件和类资源文件。在具体实现用户登录系统时，不仅基于 Struts 2.x+Guice 框架来实现，而且还通过国际化实现了语言的多元化。

第 3 篇　项目案例实战

第 23 章　在线音乐管理系统（AJAX+JSP+Struts 2.x）

随着网络时代的到来和不断发展，网络上传递的信息种类越来越多，从最初的文字信息发展到目前文字、图像、声音、视频、动画等几乎所有种类的信息。随着传递信息种类的增多，迫切需要对上传的信息进行管理。用来管理音乐的系统就是所谓的音乐管理系统。

本章将通过 AJAX+JSP+Struts 2.x 框架技术来实现一个完整的在线音乐管理系统，其中 Struts 2.x 框架的版本号为 Struts 2.0，数据库管理系统则为 MySQL。

23.1　在线音乐管理系统简述

为了让读者可以快速地理解和掌握在线音乐管理系统，在具体讲解时先按照面向不同的使用对象分成两部分：管理员模块（后台系统）和注册用户模块（前台系统）。在具体实现各个模块的相应功能时，利用 AJAX 和 JSP 技术实现该系统的相应页面，利用 Struts 2.x 框架技术实现业务逻辑。本节将以直观的方式讲解整个在线音乐管理系统。

23.1.1　在线音乐管理系统描述——后台系统

本节将以直观的方式，向读者介绍整个在线音乐管理系统中后台系统的功能。从后台主界面可以发现超级管理员可以实现如下功能：音乐管理、友情链接、用户管理、添加管理员、修改密码和关闭。

首先介绍一下超级管理员所拥护的权限，当在如图 23.1 所示的页面中输入正确的用户名和密码后，就可以直接进入后台的主界面。



图 23.1　后台登录页面

1．友情链接管理

进入后台主界面后，单击“友情链接”按钮就可以打开管理友情链接页面，如图 23.2 所示。在该页面中填写相应的信息后，单击“确定”按钮就可以实现添加友情链接功能，如图 23.3 所示。当添加完友情链接后，在管理友情链接页面的下面就会显示出友情链接信息列表。如果管理员想删除某个友情链接，可以单击友情链接信息记录中的“删除”链接，如图 23.4 所示。

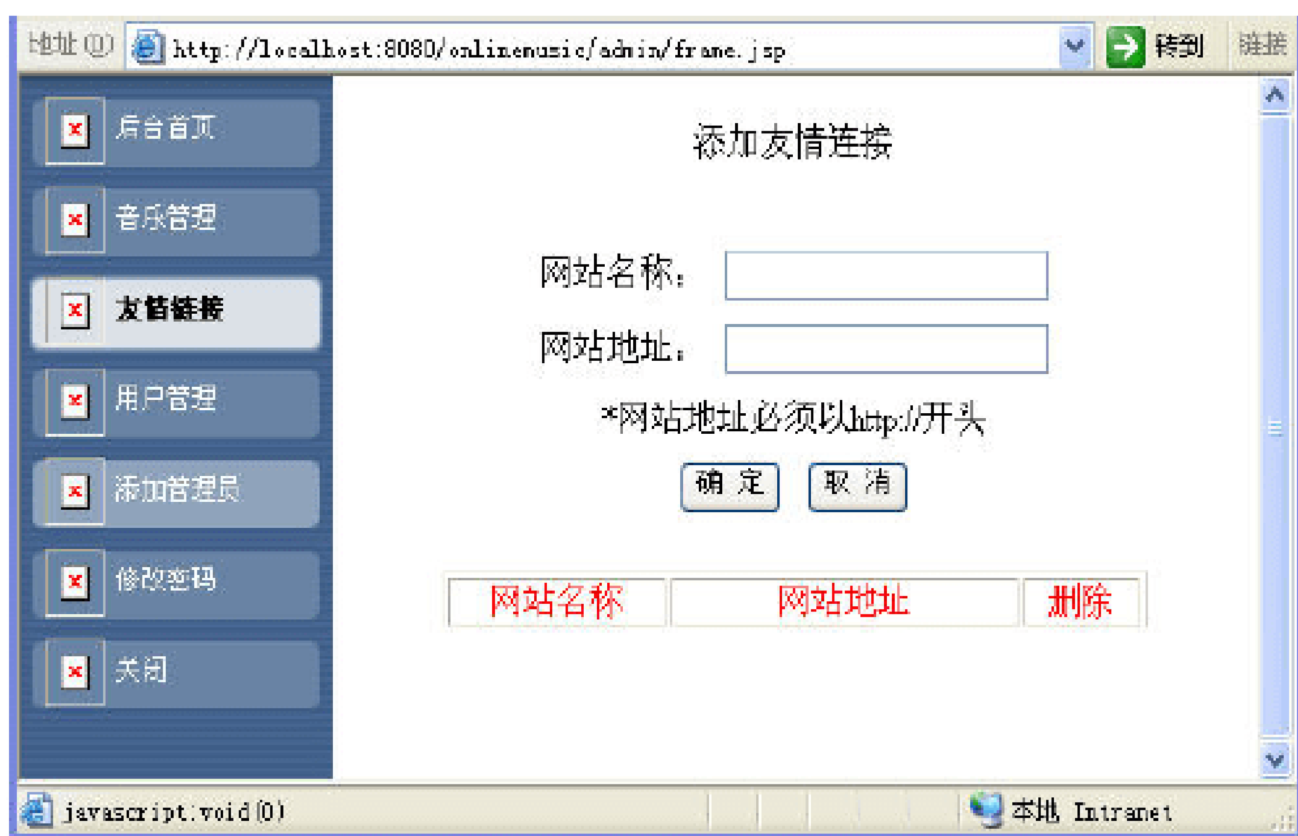


图 23.2　管理友情链接



图 23.3　添加友情链接



图 23.4　删除友情链接

当超级管理员操作完友情链接后，其他用户浏览该系统时，则会在首页的右下角显示出添加好的链接。当单击“友情链接”面板中的“百度”链接后（如图 23.5 所示），就可以转到百度的首页，如图 23.6 所示。



图 23.5　显示友情链接



图 23.6　百度首页

2. 用户管理

进入后台主界面后，单击“用户管理”按钮就可以打开用户管理页面（如图 23.7 所示），在该页面会显示出已注册用户信息的列表。如果管理员想删除某个注册用户，可以单击用户信息记录中的“删除”链接。



图 23.7　用户管理

3. 添加管理员

进入后台主界面后，单击“添加管理员”按钮就可以打开添加管理员页面如图 23.8 所示。在该页面填写相应信息后，单击“注册”按钮就可以实现添加一个新的注册用户功能，如图 23.9 所示。

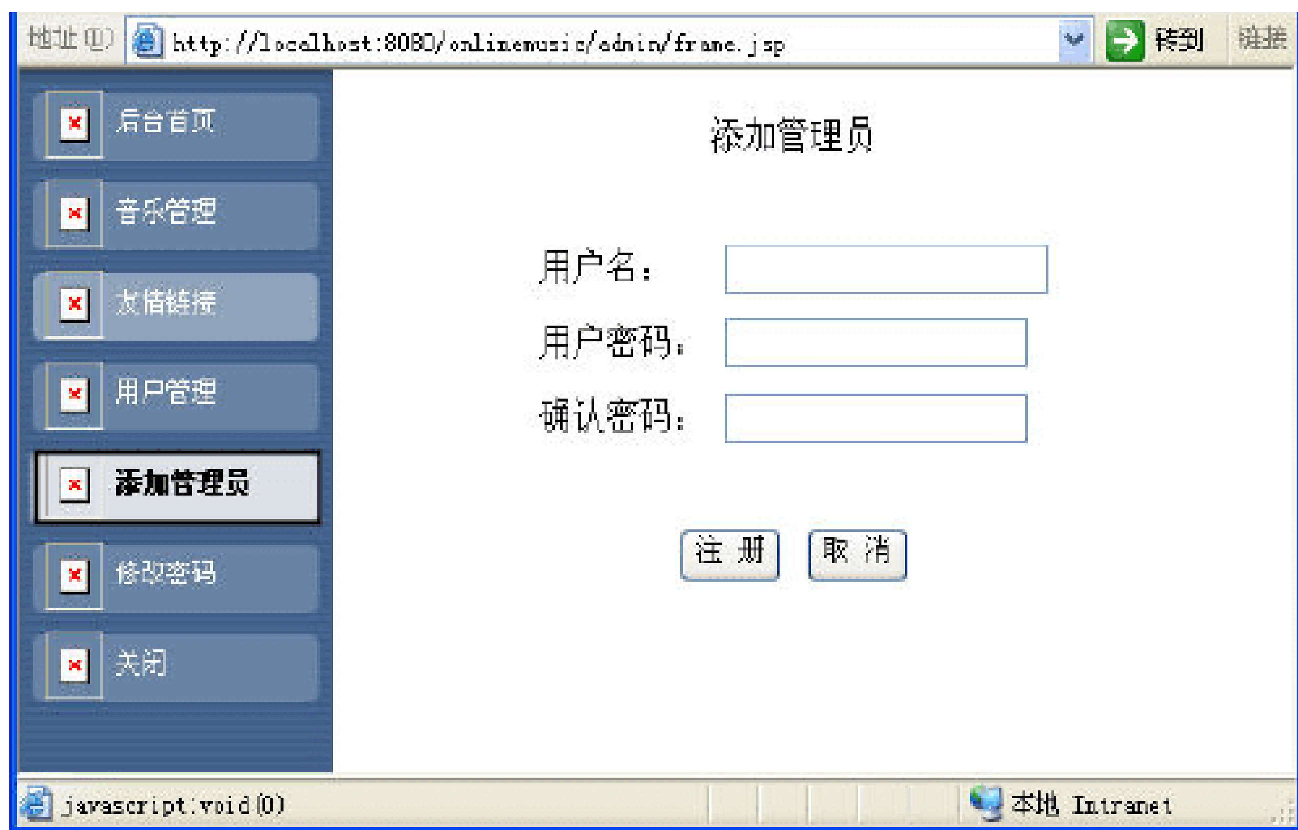


图 23.8　注册管理员

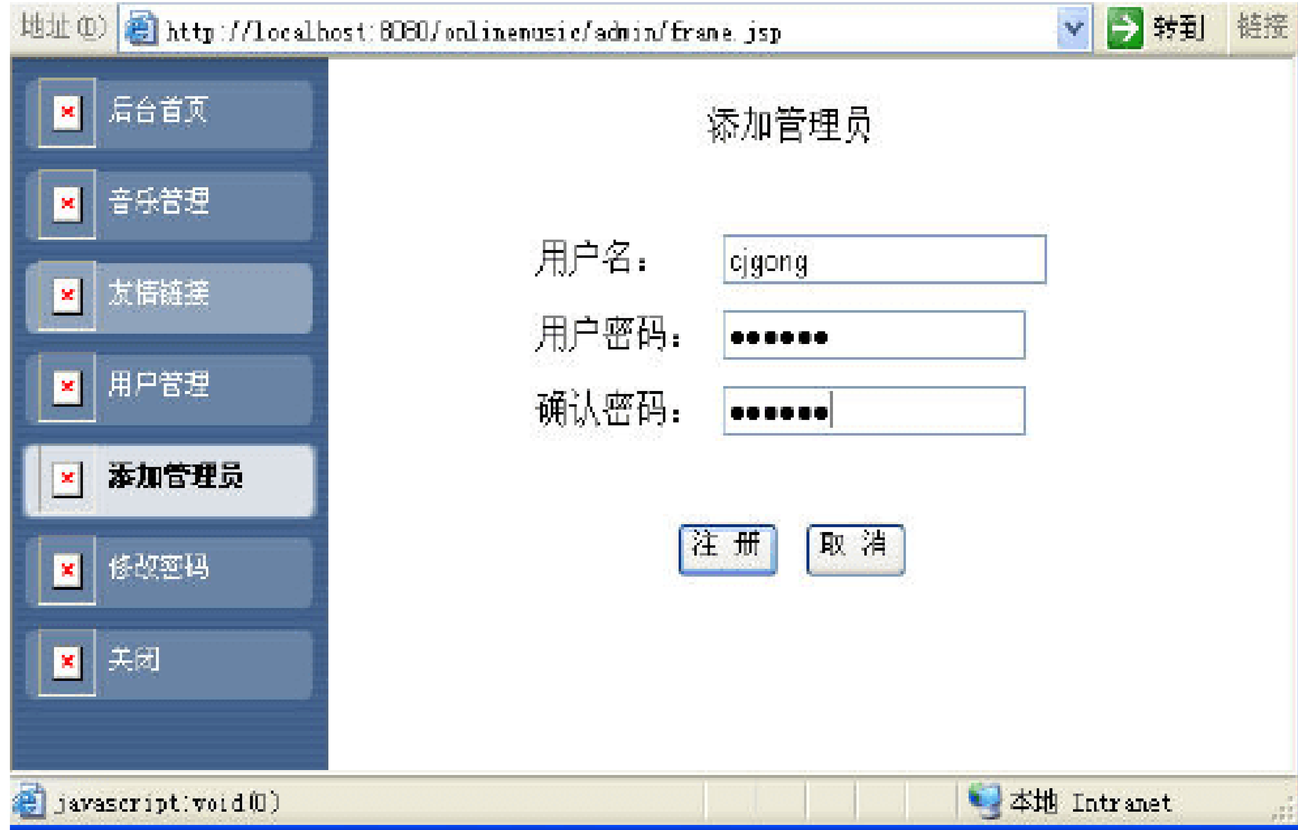


图 23.9　添加新的管理员

4. 修改密码

进入后台主界面后，单击“修改密码”按钮就可以打开修改当前管理员密码页面，如图 23.10 所示。在该页面填写相应信息后，单击“提交”按钮就可以实现修改当前管理员的密码功能，如图 23.11 所示。



图 23.10　修改密码

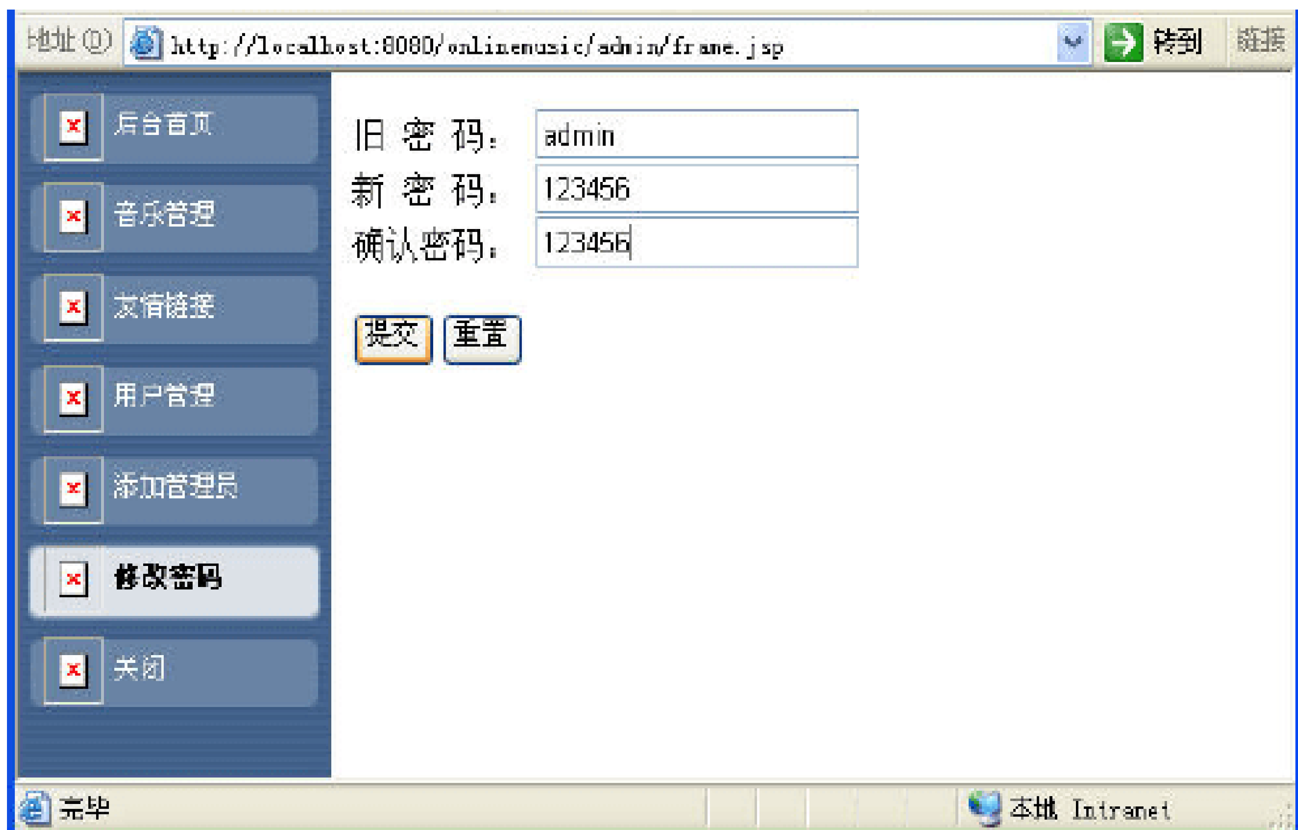


图 23.11　修改密码

23.1.2　在线音乐管理系统描述——前台系统

本节将以直观的方式，向读者介绍整个在线音乐管理系统中前台系统的功能。从前台主界面中可以发现注册用户可以实现如下功能：用户注册、用户登录、分享歌曲、填写关于音乐评论、音乐盒、点歌和试听歌曲、发送和接收短信。

当用户登录在线音乐管理系统后，首先会进入该系统的首页，如图 23.12 所示。在该页面中如果单击音乐盒、短消息、分享歌曲和播放列表导航栏，则会出现如图 23.13 所示的请登录对话框。

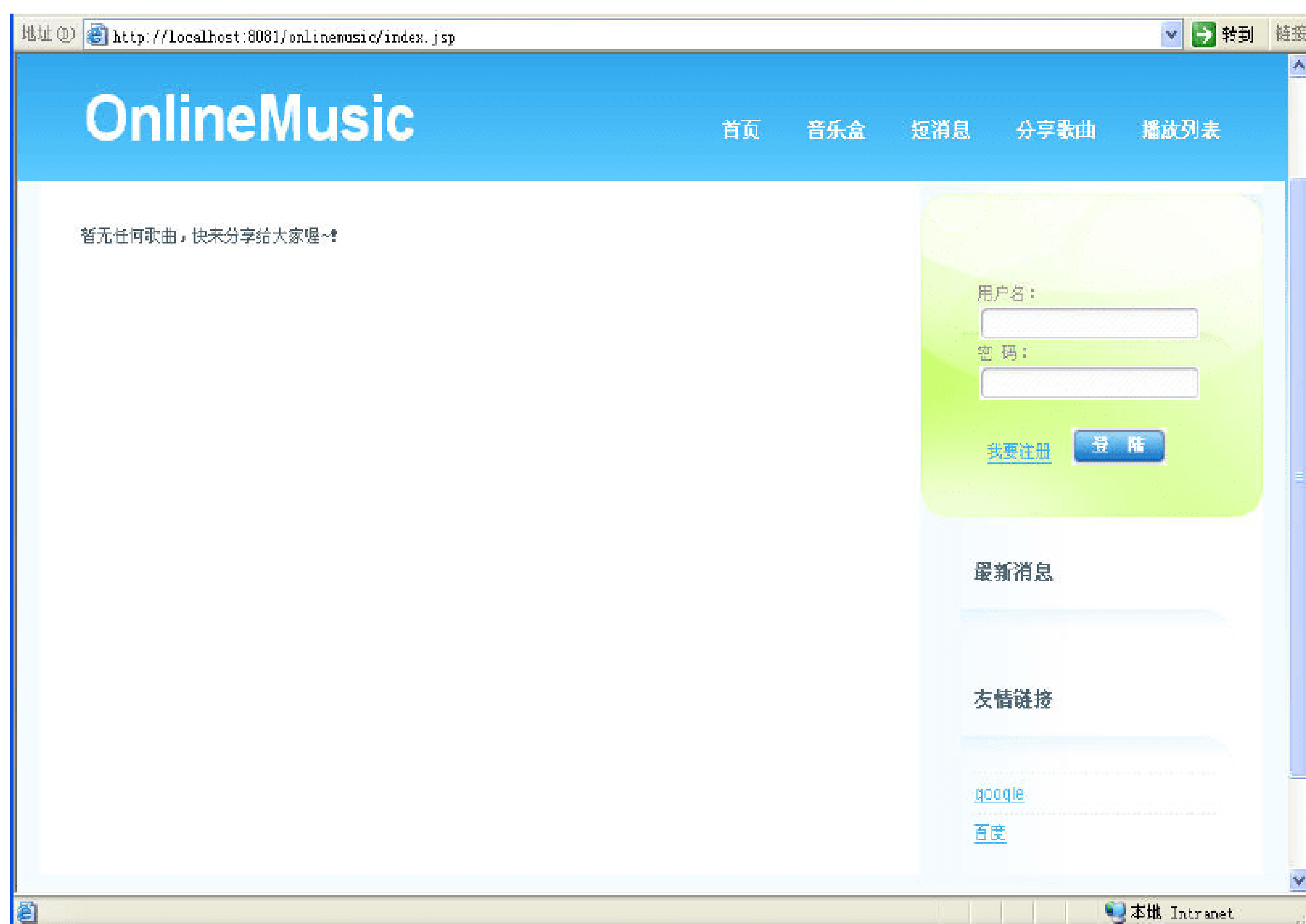


图 23.12　在线音乐管理系统首页

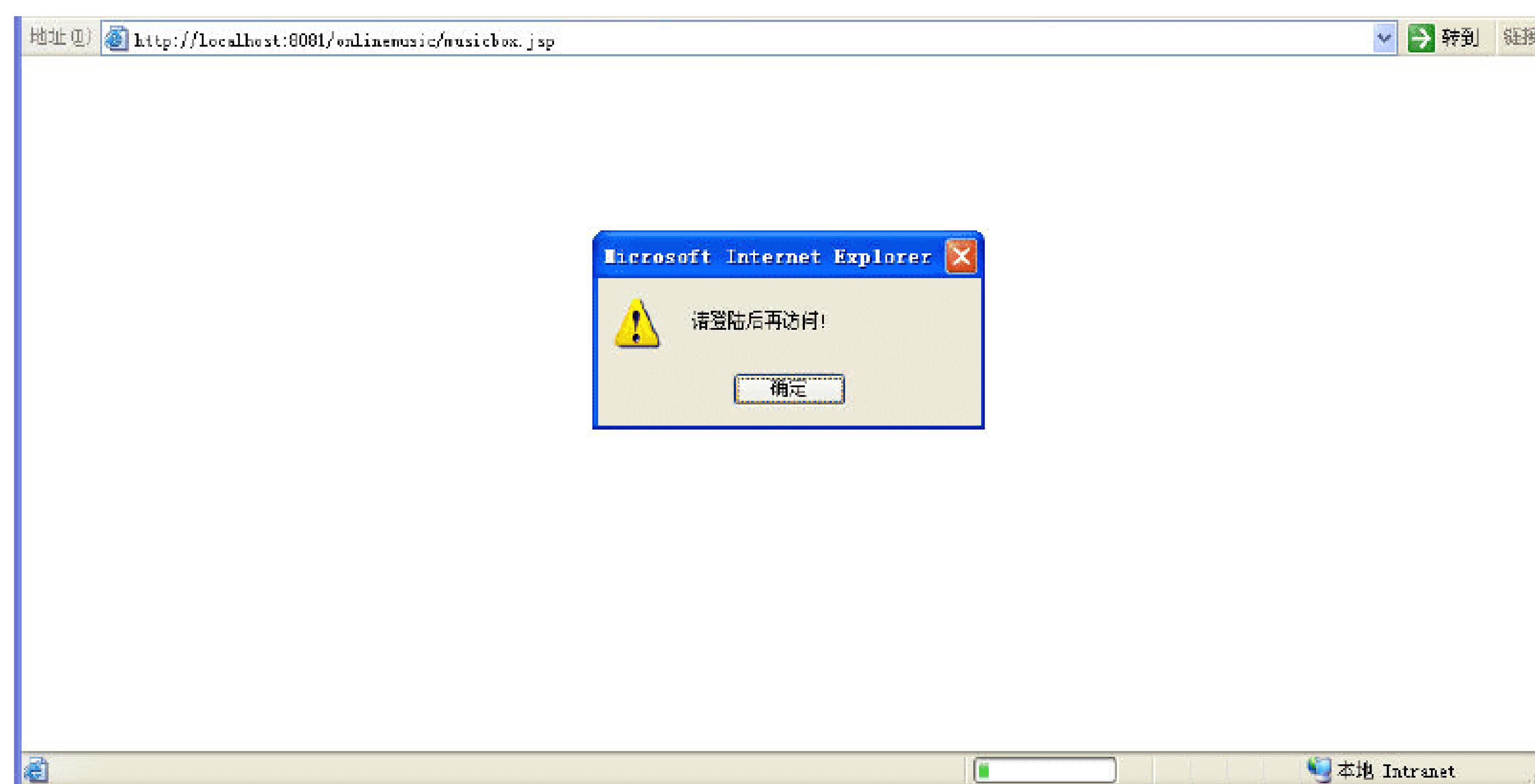


图 23.13　请登录对话框

1．用户注册

当用户浏览在线音乐管理系统时，如果想实现上传音乐等功能时，必须先注册为该系

统的用户。首先单击右边注册用户面板中的“我要注册”链接（如图 23.14 所示），就会出现“用户注册”对话框（如图 23.15 所示）。在该对话框中填写相应的信息后，单击“注册”按钮则可以实现用户注册功能，如图 23.16 所示。



图 23.14　注册用户对话框

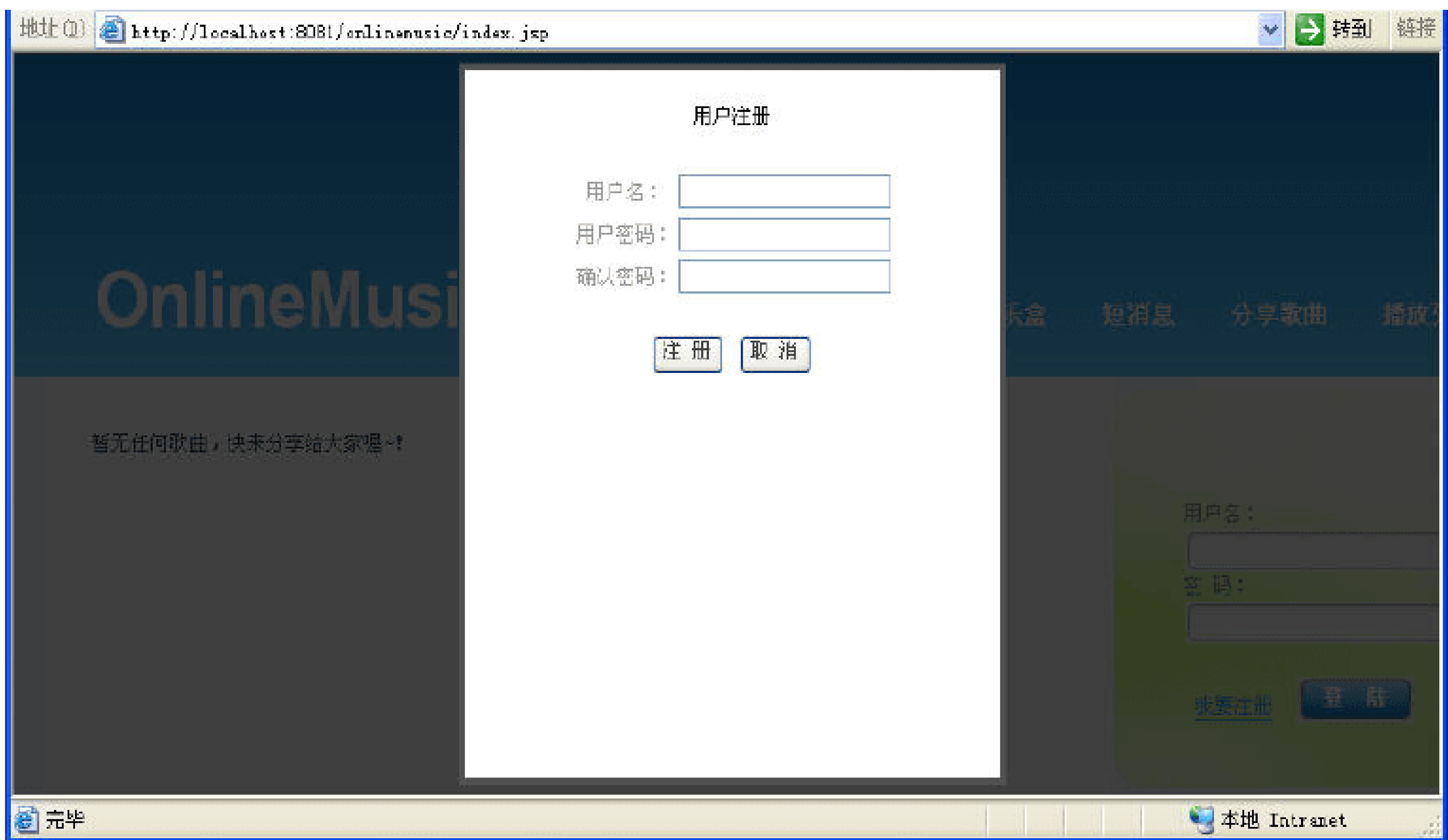


图 23.15　注册对话框



图 23.16　注册用户

当以管理员身份进入后台主界面时，单击“用户管理”按钮就可以打开用户管理页面，在该页面中会显示出所有注册用户信息，如图 23.17 所示。



图 23.17　注册用户列表

当用户在音乐系统首页中的登录面板上填写相应的用户名和密码后（如图 23.18 所示），单击“登录”按钮就可以登录音乐系统（如图 23.19 所示），在该页面中会显示出登录用户的信息。





图 23.18　用户登录　　　　　　图 23.19　用户 cjgong 主界面

2．分享歌曲

如果想共享音乐，除了可以单击“分享歌曲”导航栏外，还可以通过登录面板上的“上传音乐”链接，打开如图 23.20 所示的上传音乐对话框。在该对话框中单击“浏览”按钮选择相应的音乐文件后，单击“下一步”按钮就可以实现该音乐的上传，如图 23.21 所示。



图 23.20　上传音乐对话框



图 23.21　选择歌曲

上传完音乐后，会直接转入如图 23.22 所示的音乐描述页面。在该页面中需要填写一些关于该音乐的必要信息，具体设置如图 23.23 所示。当设置完音乐的相关信息后，单击“提交”按钮会直接进入显示音乐信息的页面，如图 23.24 所示。



图 23.22　描述音乐



图 23.23　音乐相关信息



图 23.24　完成音乐上传

当以管理员身份进入后台主界面时，单击“音乐管理”按钮就可以打开音乐管理页面，在该页面中会显示出所有上传的音乐信息，如图 23.25 所示。



图 23.25　显示上传音乐

3．填写关于音乐评论

当不以任何身份登录在线音乐管理系统后，就会出现如图 23.26 所示的页面。该页面与图 23.24 所示的页面不同，即在音乐的下面少了“添加到我的音乐盒”和“点歌”链接。单击“阅读全文”链接后，就可以出现我要留言和最近留言对话框，如图 23.27 所示。在我要留言对话框中填写如图 23.28 所示的信息，单击“提交”按钮在“最近留言”对话框中就会出现留言内容，如图 23.29 所示。



图 23.26　首页



图 23.27　我要留言对话框

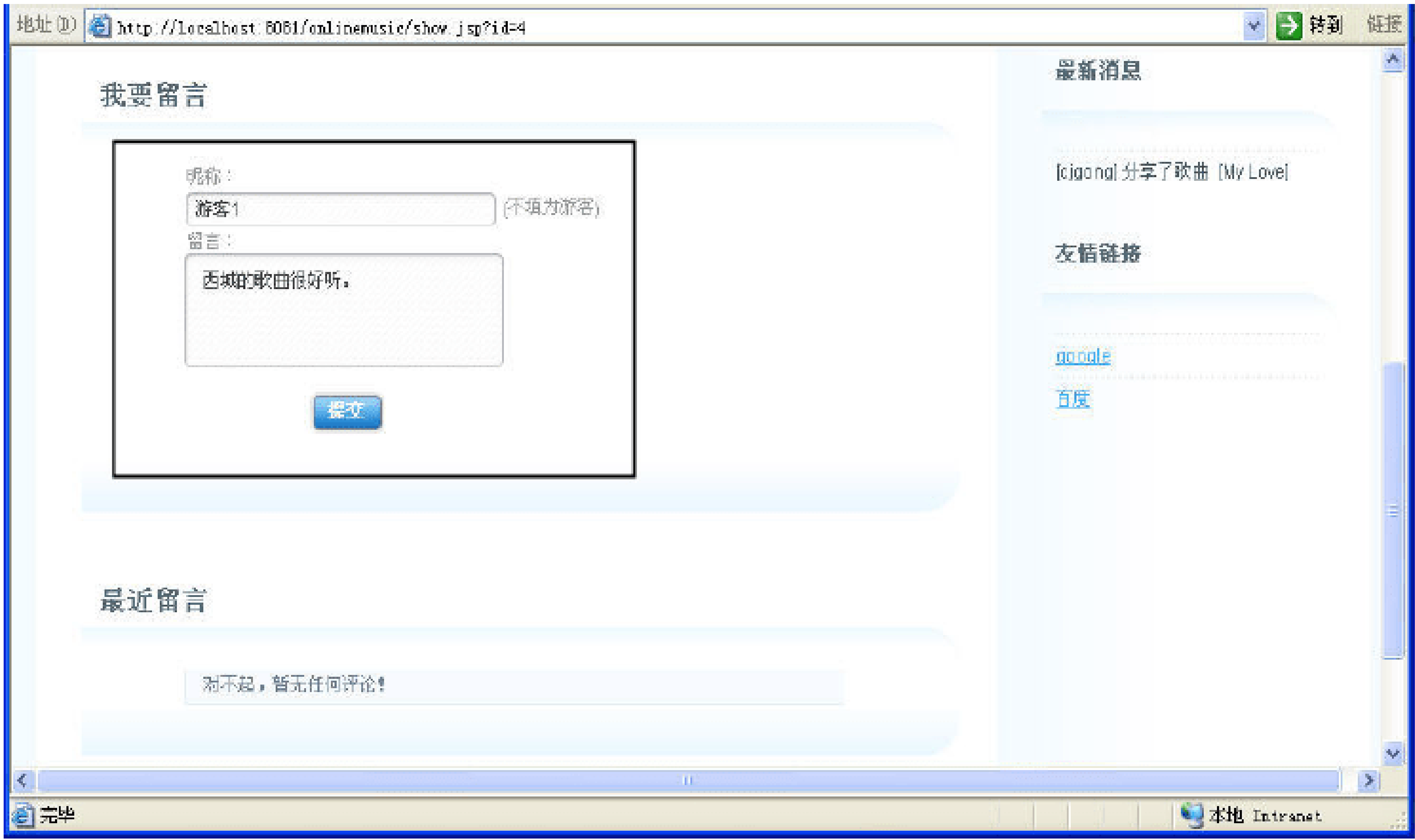


图 23.28　填写留言



图 23.29　添加留言成功

4．音乐盒管理

当以 cjgong 身份登录在线音乐管理系统后，如果这时单击导航栏中的“音乐盒”链接，

就会出现如图 23.30 所示的音乐盒页面，在该页面中没有任何歌曲。如果想添加音乐到音乐盒，可以单击“添加到我的音乐盒链接（如图 23.31 所示），就会出现如图 23.32 所示的“添加成功”对话框。



图 23.30　音乐盒



图 23.31　添加到我的音乐盒



图 23.32　添加成功对话框

当“添加到我音乐盒”成功后，单击导航栏中的“音乐盒”链接，在出现的关于“音乐盒”页面中就会出现所添加的音乐，如图 23.33 所示。

5．试听音乐

如果想试听歌曲，可以单击首页中音乐面板中的▶按钮，如图 23.34 所示。这时就会

出现显示播放进度的图标，如图 23.35 所示。如果想停止歌曲播放，直接单击⏸按钮就可以出现如图 23.36 所示的停止播放页面。



图 23.33　音乐盒

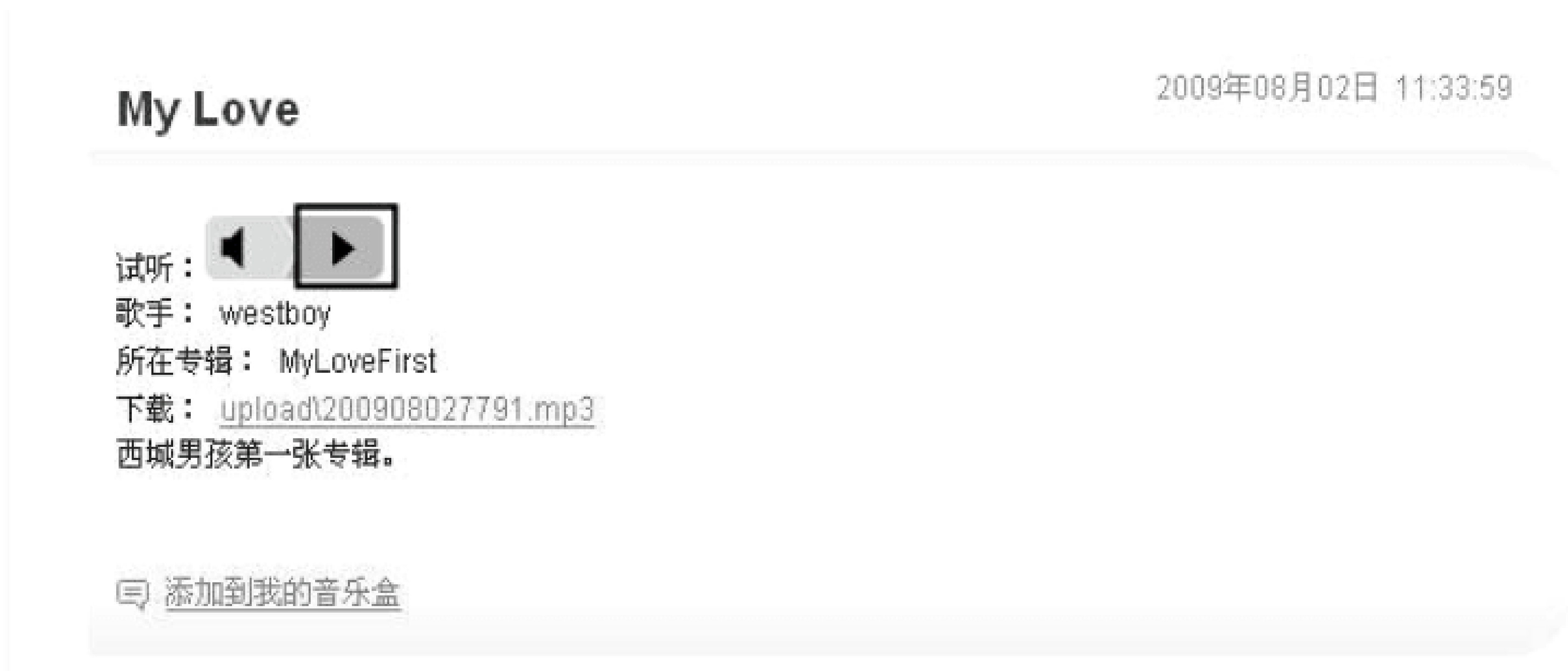


图 23.34　音乐盒面板



图 23.35　播放歌曲

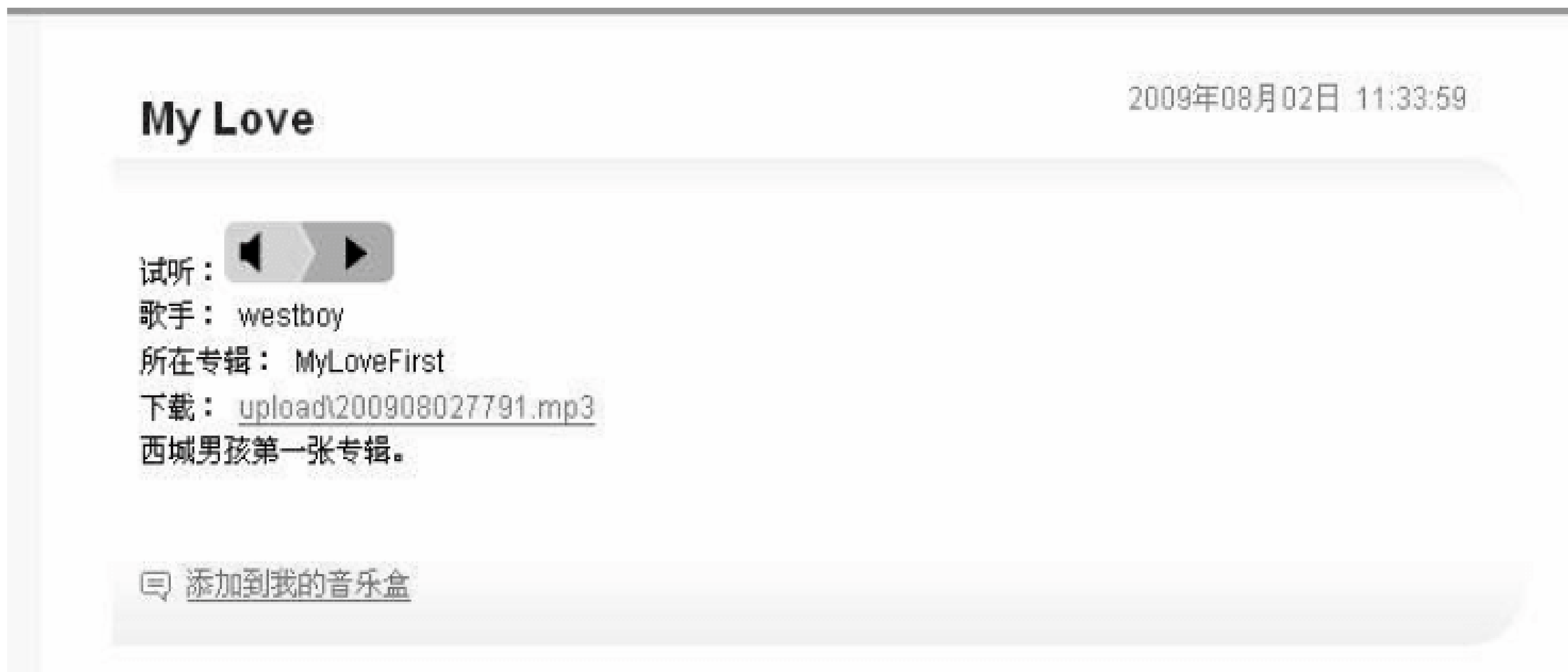


图 23.36　停止播放

6．点歌和试听歌曲

当以 cjgong1 身份登录在线音乐管理系统后，如果想给用户 cjgong 点一首歌曲，单击“点歌”链接后（如图 23.37 所示），就可以出现“点播歌曲”对话框，如图 23.38 所示。在“点播歌曲”对话框中填写如图 23.39 所示的信息，单击“确定”按钮就可以实现点播功能。



图 23.37　系统首页

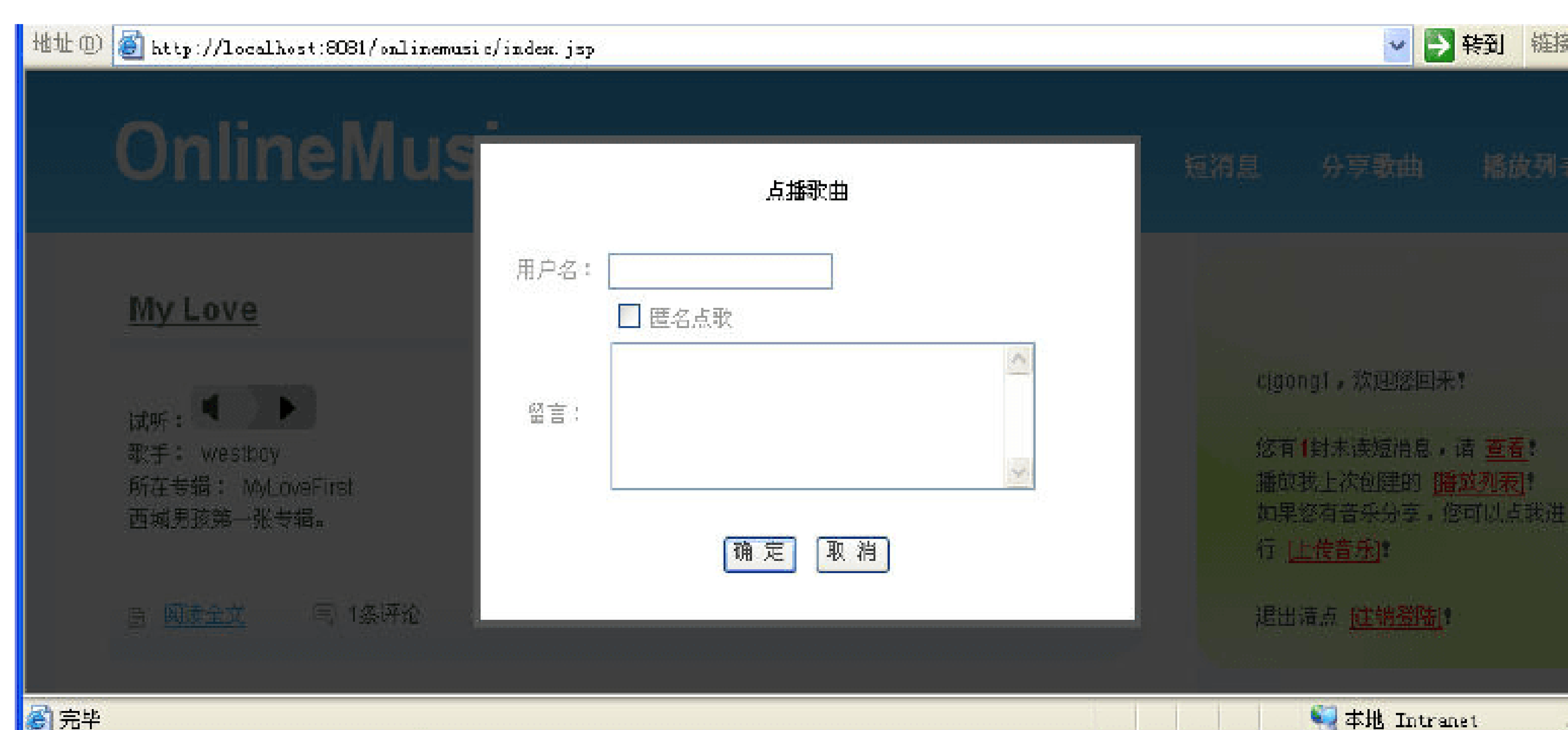


图 23.38　点歌对话框

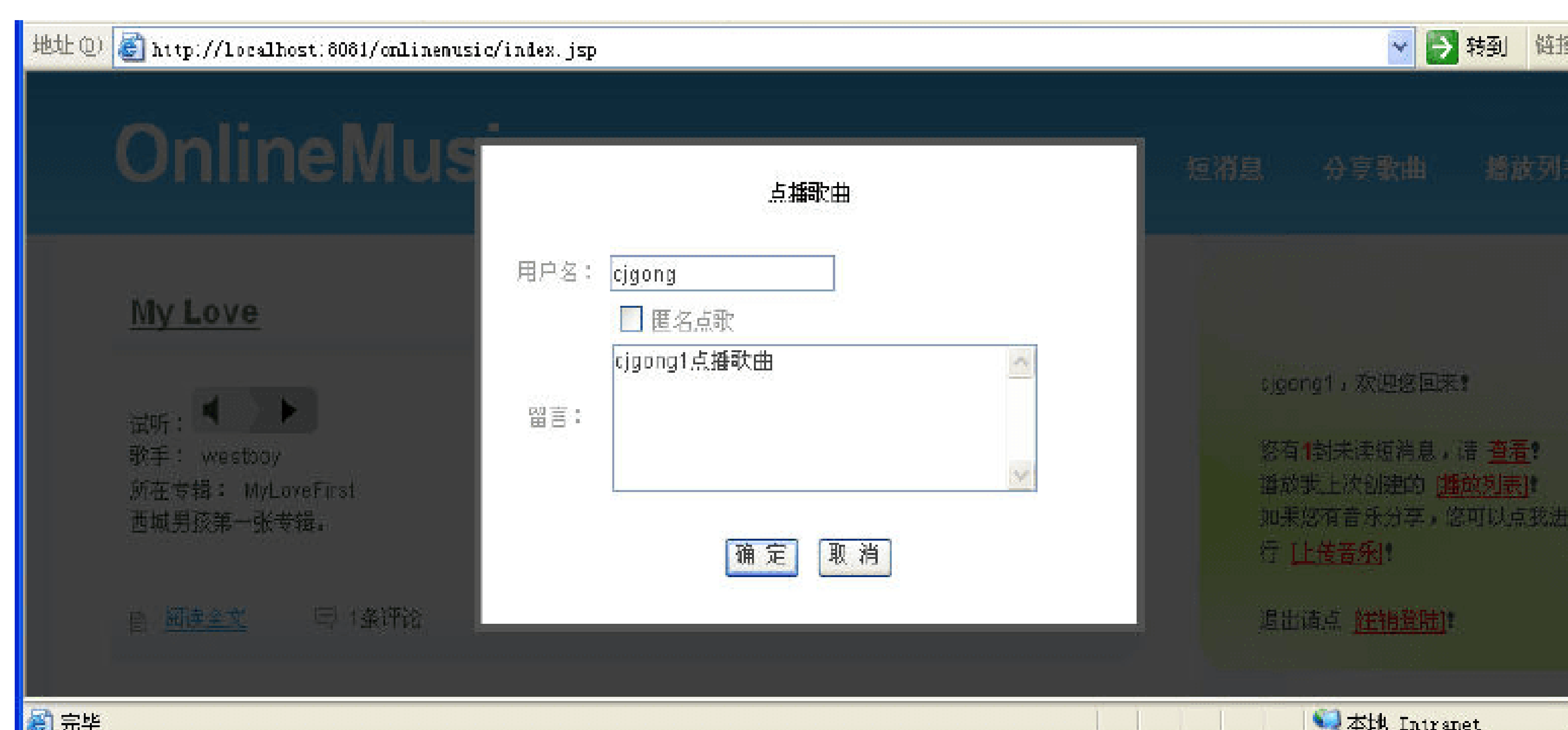


图 23.39　点歌具体信息

当用户 cjgong 登录该系统后，就会在右上角的信息框中显示出没有阅读的短信息，如图 23.40 所示。在该信息框中单击“查看”链接就可以打开如图 23.41 所示的查看点歌页

面。在该页面中单击点歌记录中标题的链接就可以直接打开关于该点歌的内容，如图 23.42 所示。



图 23.40　点歌面板



图 23.41　查看点歌



图 23.42　点听歌曲

7. 发送和接收短信消息

当注册用户进入在线音乐管理系统后，单击“短消息”导航栏（如图 23.43 所示）就可以直接进入关于短信页面，如图 23.44 所示。在该页面中显示了发给当前用户的短信和实现短信的发送功能，如果想发送短信给某个注册用户，可以通过填写相应内容并单击“提交”按钮来实现短信的发送，如图 23.45 所示。



图 23.43　单击“短信息”



图 23.44　短信页面

我的短消息！

对不起，您暂时没有任何短消息！

发送短消息！

收件人：

cjgong1

标 题：

cjgong发的短信

内容：

cjgong发的短信

提交

图 23.45　发送短信

当用户 cjgong1 登录该系统后，就会在右上角的信息框中显示出没有阅读的短信息，如图 23.46 所示。在该信息框中单击“查看”链接就可以打开如图 23.47 所示的查看短信

页面。在该页面中单击短信记录中标题的链接就可以直接打开关于该短信的内容，如图 23.48 所示。



图 23.46　短信面板



图 23.47　查看短信



图 23.48　短信的内容

至此，就完成了对在线音乐管理系统的演示。

23.2　在线音乐管理系统前期准备

本节除了将详细介绍如何设计关于在线音乐管理系统的数据库和表外，还将配置实现该系统将利用的 AJAX+JSP+Struts 2.x+MySQL 框架的环境。其中 Struts 2.x 框架的版本号为 Struts 2.0，数据库 MySQL 的版本为 MySQL 5.0。

23.2.1　设计数据库

在线音乐管理系统需要建立一个数据库并在该数据库中建立 7 张表，存放表的数据库 onlinemusic、存放超级管理员信息的表 admin、存放留言内容的表 comments、存放友情链接信息的表 link、存放留言内容的表 message、存放音乐信息的表 music、存放提示内容的

表 tip 和存放注册用户信息表 user。

1．创建数据库onlinemusic

如果想创建出数据库 onlinemusic，可以在 MySQL 的查询分析器窗口中输入如下命令：

```
CREATE DATABASE `onlinemusic`
```

2．创建表admin

如果想创建出表 admin，可以在 MySQL 的查询分析器窗口中输入相关命令。该表的具体信息如表 23.1 所示。

表 23.1　表admin信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	int	编号
Name	varchar(20)	超级管理员用户名
Pwd	varchar(32)	超级管理员密码

3．创建表comments

如果想创建出表 comments，可以在 MySQL 的查询分析器窗口中输入相关命令。该表的具体信息如表 23.2 所示。

表 23.2　表comments信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	int	编号
Value	text	留言的内容
Name	varchar(20)	留言人的昵称
music_id	int(4)	音乐编号
Time	varchar(13)	发表评论的时间

4．创建表link

如果想创建出表 link，可以在 MySQL 的查询分析器窗口中输入相关命令。该表的具体信息如表 23.3 所示。

表 23.3　表link信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	Int	编号
Value	Text	友情链接的值
Title	varchar(100)	友情链接的标题

5．创建表message

如果想创建出表 message，可以在 MySQL 的查询分析器窗口中输入相关命令。该表的

具体信息如表 23.4 所示。

表 23.4　表message信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	int	编号
From	varchar(20)	发短信的用户
To	int(4)	接受短信的用户
Title	varchar(200)	短信的标题
Value	text	短信的内容
Time	varchar(13)	发短信的时间
New	int(1)	发送短信的次数

6．创建表music

如果想创建出表 music，可以在 MySQL 的查询分析器窗口中输入相关命令。该表的具体信息如表 23.5 所示。

表 23.5　表music信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	int	编号
Title	varchar(50)	音乐的标题
singer	varchar(30)	音乐的歌曲
special	varchar(30)	音乐所属专辑
Value	text	音乐标题
Time	varchar(13)	上传时间
Click	int(5)	音乐试听次数
url	longtext	音乐的地址

7．创建表tip

如果想创建出表 tip，可以在 MySQL 的查询分析器窗口中输入相关命令。该表的具体信息如表 23.6 所示。

表 23.6　表tip信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	int	编号
Value	text	提示内容

8．创建表user

如果想创建出表 user，可以在 MySQL 的查询分析器窗口中输入相关命令。该表的具体信息如表 23.7 所示。

表 23.7　表user信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	int	编号
Name	varchar(20)	用户的编号
Pwd	varchar(32)	用户的密码
Music_box	longtext	音乐盒

至此，就完成了对在线音乐管理系统数据库和表的设计。

23.2.2　关于 Struts 2.x 框架的准备

在实现 Struts 2.x 框架、Spring 框架与 JPA 三者集成时，对于其中的 Struts 2.0 框架除了需要引入相应的 jar 文件外，还必须对 struts.xml 和 web.xml 文件做相应的配置。

1. 引入jar文件

首先引入关于 Struts 2.0 框架的核心包：struts2-core-2.0.11.jar、xwork-2.0.4.jar、ognl-2.6.11.jar、freemarker-2.3.8.jar 和 commons-logging-l.0.4.jar。然后由于该框架要与 Spring 框架整合，所以还需要 struts2-spring-plugin-2.0.8.jar。最后由于需要连接数据库 MySQL，所以还需要引入关于该数据库的驱动 MySQL-connector-java-3.1.14-bin.jar。

注意：前 6 个 jar 包在 Struts 2.0 框架的 jar 包中就可以找到，而最后一个关于数据库的驱动则必须从该数据库的官方网站上下载。

2. 修改web.xml文件

为了使在线音乐管理系统项目支持 Struts 2.0 框架，需要在 web.xml 文件中增加如代码 23.1 所示的内容。

代码 23.1　修改 web.xml 文件：web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app 2 5.xsd">
    <!--设置过滤器类-->
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <!--设置过滤器映射-->
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

3．创建struts.xml文件

为了使在线音乐管理系统项目支持 Struts 2.0 框架，需要在 onlinemusic/src 目录中创建 struts.xml 文件，该文件的内容如代码 23.2 所示。

代码 23.2　配置文件：struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!--定义全局变量-->
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
    </package>
</struts>
```

至此，就完成了对在线音乐管理系统中 Struts 2.0 框架的配置。

23.2.3　在线音乐管理系统——工具类

在一些大的项目中，总是把经常使用的方法封装成类。之所以会这样，不仅可以使程序便于阅读、修改和规范化，而且更加清晰。

DBConnection 类用来实现关于数据库的连接，具体内容如代码 23.3 所示。

代码 23.3　数据库连接类：DBConnection.java

```
public class DBConnection {
    static private String strDriver = "com.mysql.jdbc.Driver";
                                                //设置数据库驱动
    //设置数据库连接字符串
    static private String strUrl = "jdbc:mysql://localhost:3306/
    onlinemusic";
    static private String strUser = "root";     //设置用户名
    static private String strPwd = "root";      //设置用户密码
    private Connection conn = null;             //创建 Connection 变量
    private Statement stmt = null;              //创建 Statement 变量
    private PreparedStatement pstmt = null;  //创建 PreparedStatement 变量
    private ResultSet rs = null;                //创建 ResultSet 变量
    //静态块
    static {
        try {
            Class.forName(strDriver);           //加载数据库驱动
        } catch (ClassNotFoundException ex) {
            System.out.println("Error load" + strDriver);
        }
    }
    public DBConnection() {                     //无参构造函数
    }
    private Connection getConnection() {        //编写获取连接方法
        try {
```

```
            if (conn == null || conn.isClosed())
                conn = DriverManager.getConnection(strUrl, strUser, strPwd);
        } catch (Exception ex) {
            ex.printStackTrace();
            return null;
        }
        return conn;
    }
    public void close() {                                  //编写关闭方法
        try {
            if (rs != null) {
                rs.close();
                rs = null;
            }
            if (pstmt != null) {
                pstmt.close();
                pstmt = null;
            }
            if (conn != null) {
                conn.close();
                conn = null;
            }
        } catch (Exception ex) {
            System.err.println("close error:" + ex.getMessage());
        }
    }
    public ResultSet executeQuery(String sql) {            //编写查找方法
        try {
            pstmt = getConnection().prepareStatement(sql);
            rs = pstmt.executeQuery();
        } catch (SQLException ex) {
            System.err.println("query error:" + ex.getMessage());
        }
        return rs;
    }
    public boolean execute(String sql) {                   //编写执行方法
        try {
            pstmt = getConnection().prepareStatement(sql);
            if (pstmt.execute()) {
                return true;
            }
        } catch (SQLException ex) {
            System.err.println("query error:" + ex.getMessage());
            return false;
        }
        return true;
    }
    public int executeUpdate(String sql) {                 //编写更新方法
        int resultNum = 0;
        try {
            pstmt = getConnection().prepareStatement(sql);
            resultNum = pstmt.executeUpdate();
        } catch (SQLException ex) {
            System.err.println("update error:" + ex.getMessage());
        } finally {
        }
        return resultNum;
    }
}
```

【代码解析】

在上述代码中，主要通过 Java 方面的 JDBC 驱动类来实现对数据库的各种操作：获取连接方法、关闭连接方法、查找方法和更新方法。

在线音乐管理系统中不仅需要获取数据库连接的工具类外，还需要判断传递的参数是否为空的函数、实现 MD5 算法的函数等。function 类实现了该项目所需要的各种方法，具体内容如代码 23.4 所示。

代码 23.4　各种方法类：function.java

```
    ...
    public class function {
    public function() {                                    //构造函数
    }
    public static boolean isInvalid(String value) { //判断参数是否为空的方法
        return (value == null || value.length() == 0);
    }
    public static String page(int page num, int cur page, int per group,
            String base url) {                        //关于分页的方法
...
    }
    public static String page(int page num, int cur page, int per group,
                                                      //关于分页方法
            String base url,boolean noAJAX) {
...
    }
    // MD5 算法中 16 进制方法
    private final static String[] hexDigits = { "0", "1", "2", "3", "4", "5",
            "6", "7", "8", "9", "a", "b", "c", "d", "e", "f" };
    public static String byteArrayToHexString(byte[] b) {
        StringBuffer resultSb = new StringBuffer();
        for (int i = 0; i < b.length; i++) {
            resultSb.append(byteToHexString(b[i]));
        }
        return resultSb.toString();
    }
    private static String byteToHexString(byte b) {
                                                  //关于有字节型向十六进制转换
        int n = b;
        if (n < 0)
            n = 256 + n;
        int d1 = n / 16;
        int d2 = n % 16;
        return hexDigits[d1] + hexDigits[d2];
    }
    public static String MD5Encode(String origin) {    //关于 MD5 方法加密
        String resultString = null;
        try {
            resultString = new String(origin);
            MessageDigest md = MessageDigest.getInstance("MD5");
            resultString = byteArrayToHexString(md.digest(resultString
                    .getBytes()));
        } catch (Exception ex) {
        }
        return resultString;
    }
    public static String PlutoJump(String errorStr, String jumpTo) {
```

```
                                                //关于跳出窗口的方法
        String str = null;
        try {
            //关于窗口的 Script 代码
            str = "<script language='javascript'>alert('" + errorStr
                    + "');location.href='" + jumpTo + "';</script>";
        } catch (Exception e) {
            str = "<script language='javascript'>alert('" + errorStr
                    + "');location.href='" + jumpTo + "';</script>";
        }
        return str;
    }
    public static int strToInt(String str) {   //实现字符串类型向整型类型转换
        int a = 0;
        try {
            a = Integer.parseInt(str);          //实现转向功能
        } catch (NumberFormatException e) {
            a = 0;
        }
        return a;
    }
    public static String fileType(File file) { //获取文件的类型
        FileTypeMap map = FileTypeMap.getDefaultFileTypeMap();
        return map.getContentType(file);
    }
    public static byte[] readFile(String filename) throws IOException {
                                                //关于读取文件类
...
    }
}
```

注意：上述代码的各种方法，不需要程序员自己编写，因为网络上存在许多关于这些方法的资料。

至此，就完成了关于在线音乐管理系统的各种工具类。

23.3　在线音乐管理系统具体实现——超级管理员操作

为了让读者可以快速理解和掌握在线音乐管理系统，在具体讲解时按照不同的用户分成两大块：超级管理员操作和注册用户操作。

本节将详细讲解关于超级管理员的各种操作，分别为注册功能、登录功能、修改超级管理员密码功能、删除注册用户功能、删除上传音乐功能和操作友情链接功能。

23.3.1　实现超级管理员注册功能

在关于超级管理员的操作中，首先需要注册一个超级管理员，只有注册成功的超级管理员才能登录在线音乐管理系统的后台管理模块。register.java 类实现了超级管理员的注册功能，具体内容如代码 23.5 所示。

代码 23.5　超级管理员注册：register.java

```
...
public class register extends ActionSupport {
    //创建字段
    private String userName = null;
    private String userPwd = null;
    private String confirmPwd = null;
    //省略上述属性的 get()和 set()方法
...
    //编写 execute()方法
    public String execute() throws SQLException, IOException{
        //设置编码格式

    ServletActionContext.getResponse().setCharacterEncoding("GB2312");
        //获取输入流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置返回信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //判断请求的参数是否为空
        if(function.isInvalid(userName) || function.isInvalid(userPwd) ||
        function.isInvalid(confirmPwd)){
            out.println(function.PlutoJump("用户名或密码输入错误！",
            "new.jsp"));
        }
        //判断密码与确认密码是否相同
        if(!userPwd.equals(confirmPwd)){
            out.println(function.PlutoJump("两次输入的密码不一致！",
            "new.jsp"));
        }
        DBConnection conn = new DBConnection();        //获取数据库连接
        //判断注册用户名是否已经存在
        ResultSet rs = conn.executeQuery("select * from admin where name =
        '"+userName+"'");
        if(rs.next()){                                 //如果用户名已经存在
            out.println(function.PlutoJump("用户名已存在！", "new.jsp"));
        }else{
            //实现超级管理员注册
            boolean insert = conn.execute("insert into admin(name,pwd)
            values('"+userName+"','"+function.MD5Encode(userPwd)+"')");
            if(insert){                                //当注册成功时
                out.println(function.PlutoJump("注册成功，请登录！",
                "new.jsp"));
            }else{                                     //当注册失败时
                out.println(function.PlutoJump("注册失败！", "new.jsp"));
            }
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类获取 Servlet 的相关类，然后通过 Servlet

的相关类获取所要注册的超级管理员相应信息，最后通过 SQL 语句中的 insert()方法来实现注册功能。

接着在 struts.xml 文件中配置 register 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置 Action-->
        <action name="admin register" class="Pluto.admin.register">
        </action>
...
    </package>
</struts>
```

上述代码中涉及 new.jsp 页面，该页面被用来作为注册超级管理员页面，具体内容如代码 23.6 所示。

代码 23.6　注册超级管理员页面：new.jsp

```
...
    <BODY>
            <!--表单-->
            <form action="admin register.action" method="post">
                <TABLE>
                        <TD colspan="2">
                            <div align="center">添加管理员</div>
                        </TD>
                        <!--用户名输入框-->
                        <TD>用户名：</TD>
                        <INPUT type="text" name="userName" maxlength="16" />
                        <!--密码输入框-->
                        <TD>用户密码：</TD>
                        <INPUT type="password" name="userPwd" maxlength=
                        "16" />
                        <!--确认密码输入框-->
                        <TD>确认密码：</TD>
                        <INPUT type="password" name="confirmPwd" maxlength=
                        "16" />
                            <TD colSpan=2>
                                    <!--相关按钮-->
                                    <INPUT id=Login type=submit value=
                                    "注册">
                                    <INPUT type="reset" value="取 消" />
                            </TD>
                    </TBODY>
                </TABLE>
            </form>
    </BODY>
...
```

至此，就完成了超级管理员注册功能。

23.3.2　实现超级管理员登录功能

注册成功超级管理员后，可以通过该管理员的账号和密码登录在线音乐管理系统的后台管理模块。login.java 类实现了超级管理员的登录功能，具体内容如代码 23.7 所示。

代码 23.7　超级管理员登录：login.java

```
...
public class login extends ActionSupport {
    //创建字段
    private String adminName;
    private String adminPwd;
    //省略上述属性的 get()和 set()方法
...
    //编写 execute()方法
    public String execute() throws Exception {
        //设置编码格式
        ServletActionContext.getResponse().setCharacterEncoding("GB2312");
        //获取输入流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置返回信息
        HttpSession session = ServletActionContext.getRequest().
        getSession();
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //判断参数是否为空
        if (function.isInvalid(adminName) || function.isInvalid(adminPwd)) {
            out.println(function.PlutoJump("用户名或密码不能为空", "index.
            jsp"));
        } else {
            //设置 Session 对象
            session.setAttribute("PlutoAdmin", adminName);
            adminPwd = function.MD5Encode(adminPwd);  //对密码实现 MD5 编码
            DBConnection conn = new DBConnection();    //获取连接对象
            //获取执行结果
            ResultSet rs = conn
                    .executeQuery("select * from admin where name = '"
                            + session.getAttribute("PlutoAdmin").toString()
                            + "' and pwd = '" + adminPwd + "'");
            if (rs.next()) {                              //登录成功
                out
                        .println("<script language='javascript'>location.
                        href='frame.jsp';</script>");
            } else {                                      //登录失败
                session.removeAttribute("PlutoAdmin");
                out.println(function.PlutoJump("用户名密码错误", "index.
                jsp"));
            }
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类获取 Servlet 的相关类，然后通过 Servlet 的相关类获取登录用户的相应信息，最后通过 SQL 语句中的 select()方法来实现登录功能。

在 struts.xml 文件中配置 login 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 admin_login 的 Action-->
        <action name="admin_login" class="Pluto.admin.login"></action>
...
    </package>
</struts>>
```

上述代码中涉及 index.jsp 页面，该页面被用来作为登录页面，具体内容如代码 23.8 所示。

代码 23.8　管理员登录页面：index.jsp

```
...
    <BODY>
        <!--表单-->
        <form action="admin login.action" method="post">
            <TABLE>
                <TD >
                    管理员登录
                </TD>
                <!--用户名输入框-->
                <TD>用户名： </TD>
                <INPUT type="text" name="adminName" maxlength="16" />
                <!--密码输入框-->
                <TD>用户密码： </TD>
                <INPUT type="password" name="adminPwd" maxlength="16" />
                <TD colSpan=2>
                    <!--登录和取消按钮-->
                    <div align="center">
                        <INPUT id=Login type=submit value="登 录">
                        <INPUT type="reset" value="取 消" />
                    </div>
                </TD>
                </TABLE>
            </form>
    </BODY>
...
```

至此，就完成了超级管理员登录功能。

23.3.3　实现修改当前超级管理员密码功能

当超级管理员登录在线音乐管理系统的后台系统后，可以实现修改当前该超级管理员密码的功能。changepwd.java 类实现密码的修改功能，具体内容如代码 23.9 所示。

代码 23.9　改变管理员信息：changepwd.java

```
...
public class changepwd extends ActionSupport {
    //创建字段
    private String oldpwd;
    private String newpwd1;
    private String newpwd2;
    //省略上述属性的 get()和 set()方法
...
    //编写执行方法
    public String execute() throws Exception {
        //设置页面编码格式

    ServletActionContext.getResponse().setCharacterEncoding("UTF-8");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
       String adminName = ServletActionContext.getContext().getSession().
        get(
                "PlutoAdmin").toString();
        //判断参数是否为空
        if (function.isInvalid(oldpwd) || function.isInvalid(newpwd1)
                || function.isInvalid(newpwd2)) {
            out.println(function.PlutoJump("请填写密码！", "changepwd.
            jsp"));
        }
        if (newpwd1.equals(newpwd2)) {          //当新密码与确认密码相同
            DBConnection conn = new DBConnection();
            ResultSet rs = conn
                    .executeQuery("select pwd from admin where name = '"
                            + adminName + "'"); //获取当前超级管理员密码
            rs.next();
            //当输入当前超级管理员密码与旧密码相同时
            if (function.MD5Encode(oldpwd).equals(rs.getString("pwd"))) {
                //用新密码更新数据
                boolean update = conn.execute("update admin set pwd  = '"
                        + function.MD5Encode(newpwd1) + "' where name='"
                        + adminName + "'");
                if (update) {                   //更新成功
                   out.println(function.PlutoJump("修改成功！", "changepwd.
                    jsp"));
                } else {                        //更新失败
                   out.println(function.PlutoJump("修改失败！", "changepwd.
                   jsp"));
                }
            } else {
                //不具有更新密码功能
                out.println(function.PlutoJump("旧密码不对！", "changepwd.
                jsp"));
            }
        } else {
            //新密码与确认密码不相同时
            out.println(function.PlutoJump("两次输入的密码不一致！",
```

```
            "changepwd.jsp"));
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后从数据库中获取当前超级管理员密码，验证是否具有修改管理员密码。最后更新数据库中超级管理员的密码。

在 struts.xml 文件中配置 changepwd 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 admin_login Action-->
        <action name="admin login" class="Pluto.admin. changepwd">
        </action>
...
    </package>
</struts>>
```

上述代码中涉及 changepwd.jsp 页面，该页面被用来修改超级管理员密码，具体内容如代码 23.10 所示。

代码 23.10　修改超级管理员密码页面：changepwd.jsp

```
...
    <body>
        <!--表单-->
        <form id="form1" name="form1" method="post" action="admin
        changepwd.action">
            <label>
                <!--密码输入框-->
                旧 密 码:
                <input type="text" name="oldpwd" id="textfield" />
            </label>
            <label>
                <!--密码输入框-->
                新 密 码:
                <input type="text" name="newpwd1" id="textfield2" />
            </label>
            <label>
                <!--密码输入框-->
                确认密码:
                <input type="text" name="newpwd2" id="textfield3" />
            </label>
            <label>
                <!--提交和重置按钮-->
                <input type="submit" name="button" id="button" value=
                "提交" />
                <input type="reset" name="button2" id="button2" value=
                "重置" />
            </label>
        </form>
```

```
    </body>
...
```

至此，就完成了修改当前超级管理员密码功能。

23.3.4　实现删除注册用户功能

当超级管理员登录在线音乐管理系统的后台系统后，不仅可以查看到所有注册用户的信息，而且还可以删除任何一个注册用户。deluser.java 类实现删除注册用户功能，具体内容如代码 23.11 所示。

代码 23.11　删除管理员信息：deluser.java

```
...
public class deluser extends ActionSupport {
    private String id;                          //创建字段

    public String getId() {                     //配置属性 ID
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    //编写 execute()方法
    public String execute() throws Exception {

    ServletActionContext.getResponse().setCharacterEncoding("GB2312");
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        if(function.isInvalid(id)){                         //判断参数是否为空
            out.println(function.PlutoJump("出现错误！", "user.jsp"));
        }
        DBConnection conn = new DBConnection();             //获取数据库连接
        //实现删除功能
        boolean del = conn.execute("delete from user where id="+id+" limit 1");
        if(del){                                            //删除成功
            out.println(function.PlutoJump("删除成功", "user.jsp"));
        }else {                                             //删除不成功
            out.println(function.PlutoJump("删除失败", "user.jsp"));
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 Servlet 的相关类获取所要删除友情链接信息的相应信息，最后通过 SQL 语句中的 delete()方法把管理员的相应信息删除。

在 struts.xml 文件中配置 deluser 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 admin_register 的 Action-->
        <action name="admin register" class="Pluto.admin.register">
        </action>
...
    </package>
</struts>
```

上述代码中涉及 user.jsp 页面，该页面被用来作为操作注册用户页面，具体内容如代码 23.12 所示。

代码 23.12　操作注册用户页面：user.jsp

```
...
    <body>
        <table >
            <!--表格头标题 -->
            <span >用户名</span>
            <span >密码(MD5 加密)</span>
            <span >删除</span>
            <%
                //查询所有的注册用户
                ResultSet rs = conn
                            .executeQuery("select * from user order by id
                            DESC");
                    while (rs.next()) {             //遍历所有的用户变量 rs
                        String id = rs.getString("id");
                                                    //获取用户的编号 ID
                        String name = rs.getString("name");
                                                    //获取用户的名字
                        String pwd = rs.getString("pwd");  //获取用户的密码
                        //输出用户的相应信息
                        out.println("<tr align=\"center\">");
                        out.println("<td>" + name + "</td>");
                        out.println("<td>" + pwd + "</td>");
                        out.println("<td><a href=\"admin deluser.action?
                        id=" + id
                                + "\">删除</a></td>");      //删除操作
                        out.println("</tr>");
                    }
                } else {
            %>
        </table>
    </body>
...
```

至此，就完成了实现删除注册用户功能。

23.3.5　实现删除上传音乐功能

当超级管理员登录在线音乐管理系统的后台系统后，不仅可以查看到所有上传音乐的信息，而且还可以删除任何一个上传音乐。delmusic.java 类实现删除上传音乐功能，具体

内容如代码 23.13 所示。

代码 23.13　删除上传音乐：delmusic.java

```
...
public class delmusic extends ActionSupport {
    private String id;                          //创建字段
    public String getId() {                     //配置属性 ID
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    //编写 execute()方法
    public String execute() throws Exception {
        //设置页面编码格式

    ServletActionContext.getResponse().setCharacterEncoding("UTF-8");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //判断参数是否为空
        if (function.isInvalid(id)) {
            out.println(function.PlutoJump("出现错误！", "music.jsp"));
        }
        DBConnection conn = new DBConnection();        //获取数据库连接
        //实现删除功能
        boolean del = conn.execute("delete from music where id=" + id
                + " limit 1");
        if (del) {                                      //删除成功
            out.println(function.PlutoJump("删除成功", "music.jsp"));
        } else {                                        //删除失败
            out.println(function.PlutoJump("删除失败", "music.jsp"));
        }
        return null;
    }
}
```

注意：上述代码与 23.3.3 节中实现“删除管理员信息”功能的 deluser.java 代码基本相同。

在 struts.xml 文件中配置 delmusic 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 admin register 的 Action-->
        <action name="admin register" class="Pluto.admin. delmusic ">
        </action>
...
    </package>
</struts>
```

上述代码中涉及 music.jsp 页面，该页面被用来作为操作上传音乐页面，具体内容如代码 23.14 所示。

代码 23.14　操作上传音乐页面：music.jsp

```
...
    <body>
        <table >
            <!--表格头标题-->
            <span >标题（点击进入详细信息）</span>
            <span >删除</span>
            <%
                //查询相应的数据
                ResultSet rs = conn
                            .executeQuery("select * from music order by id
                            DESC");
                    //遍历查询结果
                    while (rs.next()) {
                        String id = rs.getString("id");     //获取音乐的 ID
                        String title = rs.getString("title");
                                                            //获取音乐的标题
                        //输出音乐的相应信息
                        out.println("<tr align=\"center\">");
                        out.println("<td><a href=\"../show.jsp?id=" + id
                                + "\" target=\"_blank\">" + title +
                                "</a></td>");
                        out.println("<td><a href=\"admin_delmusic.action?
                        id=" + id
                                + "\">删除</a></td>");       //实现删除功能
                        out.println("</tr>");
                    }
            %>
        </table>
    </body>
...
```

至此，就完成了实现删除上传音乐的功能。

23.3.6　操作友情链接

当超级管理员登录在线音乐管理系统的后台系统后，不仅可以添加友情链接的信息，而且还可以删除任何已存在的超级链接。link.java 类实现添加友情链接的功能，具体内容如代码 23.15 所示。

代码 23.15　添加友情链接：link.java

```
...
public class link extends ActionSupport {
    //创建字段
    private String title;
    private String value;
    //省略配置上述属性的 get()和 set()方法
    ...
    //编写 execute()方法
```

```
    public String execute() throws Exception {
        //设置页面编码格式
        ServletActionContext.getResponse().setCharacterEncoding("UTF-8");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //判断参数是否为空
        if (function.isInvalid(title) || function.isInvalid(value)) {
            out.println(function.PlutoJump("请填入网站名称和地址！", "link.
            jsp"));
        }
        DBConnection conn = new DBConnection();            //获取连接
        //实现插入功能
        boolean insert = conn.execute("insert into link(title,value)
        values('"
                + title + "','" + value + "')");
        if(insert){                                         //插入成功
            out.println(function.PlutoJump("添加成功！", "link.jsp"));
        }else{                                              //插入失败
            out.println(function.PlutoJump("添加失败！", "link.jsp"));
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 SQL 语句中的 insert()方法把“友情链接”的相应信息插入数据库。

在 struts.xml 文件中配置 link 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 admin_register 的 Action-->
        <action name="admin register" class="Pluto.admin.link"></action>
...
    </package>
</struts>
```

delink.java 类实现删除友情链接的功能，具体内容如代码 23.16 所示。

代码 23.16　删除友情链接信息：dellink.java

```
...
public class dellink extends ActionSupport {
    private String id;                            //创建字段
    public String getId() {                       //配置属性 ID
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
```

```
    //编写 execute()方法
    public String execute() throws Exception {
        ServletActionContext.getResponse().setCharacterEncoding("GB2312");
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        if (function.isInvalid(id)) {              //判断参数是否为空
            out.println(function.PlutoJump("出现错误！", "link.jsp"));
        }
        DBConnection conn = new DBConnection();    //获取数据库连接
        //实现删除功能
        boolean del = conn.execute("delete from link where id=" + id
                + " limit 1");
        if (del) {                                 //删除功能
            out.println(function.PlutoJump("删除成功", "link.jsp"));
        } else {                                   //删除失败
            out.println(function.PlutoJump("删除失败", "link.jsp"));
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 Servlet 的相关类获取所要删除友情链接信息的相应信息，最后通过 SQL 语句中的 delete() 方法把“友情链接”的相应信息删除。

在 struts.xml 文件中配置 dellink 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名 admin_register 的 Action-->
        <action name="admin register" class="Pluto.admin. dellink">
        </action>
...
    </package>
</struts>
```

在线音乐管理系统中的该模块功能中涉及 link.jsp 页面，该页面被用来作为操作友情链接页面，具体内容如代码 23.17 所示。

代码 23.17　操作上传音乐页面：link.jsp

```
...
    <BODY>
        <!--表单-->
        <form action="admin link.action" method="post">
            <TABLE>
                <TD>添加友情链接</TD>          <!--标题-->
                <TD>网站名称：</TD>            <!--网站名称输入框-->
                <INPUT type="text" name="title" maxlength="16" />
                <TD>网站地址：</TD>            <!--网站地址输入框-->
```

```
                <INPUT type="text" name="value" />
                <TD>*网站地址必须以 http://开头</TD>
                <TD >
                    <!--确定和取消按钮-->
                    <INPUT id=Login type=submit value="确 定">
                    <INPUT type="reset" value="取 消" />
                </TD>
            </TABLE>
        </form>
        <table>
            <td>
                <!--表格头表头-->
                <span>网站名称</span>
                <span>删除</span>
                <span>网站名称</span>
            </td>
            <%
                //实现查询
                ResultSet rs = conn
                            .executeQuery("select * from link order by id
                            DESC");
                    //遍历结果
                    while (rs.next()) {
                        String id = rs.getString("id"); //获取网站的 ID 号
                        String title = rs.getString("title");
                                                        //获取网站的标题
                        String value = rs.getString("value");
                                                        //获取网站的地址
                        //输出网站的相关信息
                        out.println("<tr align=\"center\">");
                        out.println("<td>" + title + "</td>");
                        out.println("<td>" + value + "</td>");
                        out.println("<td><a href=\"admin dellink.action?
                        id=" + id
                            + "\">删除</a></td>");
                        out.println("</tr>");
                    }
                } else {
            %>
        </table>
    </BODY>
...
```

至此，就完成了操作友情链接功能。

23.4　在线音乐管理系统具体实现——注册用户操作

为了让读者可以快速理解和掌握在线音乐管理系统，在具体讲解时按照不同的用户分成两大块：超级管理员操作和注册用户操作。

本节将详细讲解关于注册用户的各种操作，分别为实现用户的注册、登录和退出，实现音乐的上传和音乐信息更新、添加用户评论，实现音乐盒、短信的发送和接收，以及歌曲的点歌功能。

23.4.1　实现用户注册功能

在关于注册用户的操作中，首先需要注册成为该在线音乐管理系统的用户，只有注册成功的用户才能登录在线音乐管理系统。register.java 类实现了用户的注册功能，具体内容如代码 23.18 所示。

代码 23.18　用户注册功能：register.java

```
public class register extends ActionSupport {
    //创建字段
    private String userName = null;
    private String userPwd = null;
    private String confirmPwd = null;
    //省略上述属性的 get()和 set()方法
...
    //编写 execute()方法
    public String execute() throws SQLException, IOException{
        //设置页面编码格式

    ServletActionContext.getResponse().setCharacterEncoding("UTF-8");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        if(function.isInvalid(userName) || function.isInvalid(userPwd) ||
        function.isInvalid(confirmPwd)){
            out.println(function.PlutoJump("用户名或密码输入错误！", "index.
            jsp"));
        }
        if(!userPwd.equals(confirmPwd)){
            out.println(function.PlutoJump("两次输入的密码不一致！", "index.
            jsp"));
        }
        DBConnection conn = new DBConnection();
        ResultSet rs = conn.executeQuery("select * from user where name =
        '"+userName+"'");
        if(rs.next()){
            out.println(function.PlutoJump("用户名已存在！", "index.jsp"));
        }else{
            boolean insert = conn.execute("insert into user(name,pwd)
            values('"+userName+"','"+function.MD5Encode(userPwd)+"')");
            if(insert){
                out.println(function.PlutoJump("注册成功，请登录！", "index.
                jsp"));
            }else{
                out.println(function.PlutoJump("注册失败！", "index.jsp"));
            }
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 Servlet 的相关类获注册用户的相应信息，最后通过 SQL 语句中的 insert()方法实现用户的注册功能。

在 struts.xml 文件中配置 register 类。

```
...
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 register 的 Action-->
        <action name="register" class="Pluto.register"></action>
...
    </package>
</struts>
```

在线音乐管理系统中的该模块功能中涉及 register.jsp 页面，该页面被用来作为在线音乐管理系统的注册页面，具体内容如代码 23.19 所示。

代码 23.19　注册页面：register.jsp

```
...
    <BODY>
        <!--表单-->
        <form action="register.action" method="post">
            <TABLE>
                <TD >
                    用户注册
                </TD>
                <TD>
                    <!--用户输入框-->
                    用户名:
                    <INPUT type="text" name="userName" maxlength="16" />
                </TD>
                <TD>
                    <!--密码输入框-->
                    用户密码:
                    <INPUT type="password" name="userPwd" maxlength="16" />
                </TD
                <TD>
                    确认密码:
                    <INPUT type="password" name="confirmPwd" maxlength="16"/>
                </TD
                <TD>
                    <!--注册和取消按钮-->
                    <INPUT id=Login type=submit value="注 册">
                    <INPUT type="reset" onclick=tb_remove(); value="取 消" />
                </TD
                    </TBODY>
            </TABLE>
        </form>
    </BODY>
...
```

至此，就完成了用户注册功能。

23.4.2　实现注册用户登录和退出功能

在关于注册用户的操作中，只有注册成功的用户才能在登录后使用该系统，login.java 类实现了用户的登录功能，具体内容如代码 23.20 所示。

代码 23.20　用户登录功能：login.java

```
...
public class login extends ActionSupport {
    //创建字段
    private String userName;
    private String userPwd;
    //省略上述属性的 get()和 set()方法
...
    //编写 execute()方法
    public String execute() throws IOException, SQLException {
        //设置页面编码格式
        ServletActionContext.getResponse().setCharacterEncoding("UTF-8");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //判断参数是否为空
        if (function.isInvalid(userName) || function.isInvalid(userPwd)) {
            out.println(function.PlutoJump("用户名或密码不能为空", "index.
            jsp"));
        } else {
            session.setAttribute("PlutoUser", userName);   //设置 Session
            userPwd = function.MD5Encode(userPwd);          //转换成 MD5 编码
            DBConnection conn = new DBConnection();         //获取数据库连接
            ResultSet rs = conn
                    .executeQuery("select * from user where name = '"
                            + session.getAttribute("PlutoUser").toString()
                            + "' and pwd = '" + userPwd + "'");
                                                            //获取执行结果
            if (rs.next()) {                                //登录成功
                out
                        .println("<script language='javascript'>location.
                        href='index.jsp';</script>");
            } else {                                        //登录失败
                session.removeAttribute("PlutoUser");
                out.println(function.PlutoJump("用户名密码错误", "index.
                jsp"));
            }
        }
        return null;
    }
};
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 Servlet 的相关类获取所要注册用户的相应信息，最后通过 SQL 语句中的 select()方法实现登录功能。

在 struts.xml 文件中配置 login 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 login 的 Action-->
        <action name="login" class="Pluto.login"></action>
...
    </package>
</struts>
```

Logout.java 类实现了用户的退出功能，具体内容如代码 23.21 所示。

代码 23.21　用户退出功能：logout.java

```
public class logout extends ActionSupport {
    //编写 execute()方法
    public String execute() throws IOException{
        //设置页面编码格式
        ServletActionContext.getResponse().setCharacterEncoding("UTF-8");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //移除 Session 对象
        session.removeAttribute("PlutoUser");
        out.println(function.PlutoJump("注销成功", "index.jsp"));
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 Servlet 的相关类中 Session 对象实现用户的推出功能。

在 struts.xml 文件中配置 logout 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 logout 的 Action-->
        <action name="logout" class="Pluto.logout"></action>
...
    </package>
</struts>
```

在线音乐管理系统中的该模块功能中涉及 index.jsp 页面，在该页面中可以实现用户的登录和退出功能，具体内容如代码 23.22 所示。

代码 23.22　首页：index.jsp

```
...
    <body>
...
        <%
            //判断 PlutoUser 的值
            if (session.getAttribute("PlutoUser") == null) {
        %>
        <!--表单 -->
        <form action="login.action" method="post" class="niceform">
            <!--用户名输入框 -->
            <label for="textinput">
                  用户名:
            </label><br />
            <input type="text" id="textinput" name="userName" size="15"
                maxlength="16" /><br />
            <!--密码输入框 -->
            <label for="passwordinput">
                  密 码:
            </label><br />
            <input type="password" id="passwordinput" name="userPwd"
                size="15" maxlength="16" />
            <br /><br />
            <!--提交按钮 -->
            <a href="register.jsp?height=175&width=300&modal=true"
                class="thickbox" title="我要注册">我要注册</a>
            <input type="submit" value="登  录" />
        </form>
        <%
            } else {
        %><p>
            <%=session.getAttribute("PlutoUser").toString()%>，欢迎您回来!
            </p>
        <%
            //获取 userName 的值
            String userName = session.getAttribute("PlutoUser").
            toString();
                //执行结果
                ResultSet user rs = conn
                        .executeQuery("select id from user where name = '"
                                + userName + "'");
                user_rs.next();
                String id = user_rs.getString("id");  //获取 id 的值
                //执行结果
                ResultSet message rs = conn
                .executeQuery("select count(id) as count from message where
                'to' ="
                                + id + " and `new` = 1");
                message_rs.next();
                int myMessage = message rs.getInt("count");
        %>
            <!--退出链接 -->
            <P>退出请点
```

```
            <a href="logout.action" style="color: #FF0000">[注销登录]</a>!
            </p>
...
    </body>
...
```

至此，就完成了用户登录和退出功能。

23.4.3　实现在线音乐上传

在关于注册用户的操作中，还需要实现上传音乐文件功能。同时注意只有登录成功的注册用户才能实现音乐文件的上传。uploadmusic.java 类实现了音乐上传功能，具体内容如代码 23.23 所示。

代码 23.23　音乐在线上传：uploadmusic.java

```
public class uploadmusic extends ActionSupport {
    //创建字段
    private File upload;
    private String uploadContentType;
    private String uploadFileName;
    private String savePath;
    //省略上述属性的 get()和 set()方法
...
    //编写 execute()方法
    public String execute() throws Exception {
        //设置页面编码格式

    ServletActionContext.getResponse().setCharacterEncoding("UTF-8");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //生成文件名
        String fileType = getUploadFileName().substring(getUploadFile-
        Name().lastIndexOf("."));
        SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd");
                                                    //设置时间格式
        Date dt = new Date();                       //获取当前时间
        Random rd = new Random();                   //随机变量
        setUploadFileName(sdf.format(dt) + rd.nextInt(9999) + fileType);
                                                //生成上传文件的名字
        if ("audio/mpeg".equals(getUploadContentType())) { //判断音乐类型
            //获取文件输出流
            FileOutputStream fos = new FileOutputStream(getSavePath() +
            "\\"
                    + getUploadFileName());
            //获取文件输入流
            FileInputStream fis = new FileInputStream(getUpload());
            byte[] buffer = new byte[10240];            //创建字节数组
            int len = 0;
            while ((len = fis.read(buffer)) > 0) {      //实现文件的上传
```

```
                fos.write(buffer, 0, len);
            }
            //创建文件路径属性
            String filePath = "upload\\\\" + getUploadFileName();
            out.println(function.PlutoJump("上传成功，请认真填写歌曲内容！",
            "upload.jsp?path=" + filePath));
        } else {                                    //当上传文件失败
            out.println(function.PlutoJump("文件类型必须为MP3！",
            "uploadmusic.jsp"));
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 Java API 中的 File 类获取文件流和上传文件的时间，最后通过 Response 对象输出上传相关的信息。

在 struts.xml 文件中配置 uploadmusic 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为uploadmusic 的Action-->
        <action name="uploadmusic" class="Pluto.uploadmusic">
            <param name="savePath">/upload</param>
...
        </action>
    </package>
</struts>
```

在线音乐管理系统中的该模块功能中涉及 uploadmusic.jsp 页面，该页面被用来作为在线音乐管理系统的上传音乐页面，具体内容如代码 23.24 所示。

代码 23.24　上传音乐页面：uploadmusic.jsp

```
...
    <body>
...
        <h2 class="title">
            上传音乐第一步（上传音乐）
        </h2>
        <!--表单 -->
        <form id="form1" name="form1" method="post" enctype="multipart/
        form-data" action="uploadmusic.action" class="niceform">
            <table>
                <!--文件上传元素 -->
                <td>
                    请上传歌曲：
                    <input type="file" name="upload" id="fileField" />
                </td>
                <td>
                    <!--下一步按钮 -->
                    <input type="submit" name="button" id="button" value=
                    "下一步" />
```

```
                </td>
            </table>
        </form>
...
    </body>
...
```

至此，就完成了在线音乐上传功能。

23.4.4　实现音乐信息更新

在关于注册用户的操作中，还需要实现更新音乐信息功能。同时注意只有登录成功的注册用户才能实现音乐文件信息的更新。upload.java 类用来实现音乐信息的更新，具体内容如代码 23.25 所示。

代码 23.25　实现音乐信息更新：upload.java

```
public class upload extends ActionSupport {
    //创建字段
    private String title;
    private String singer;
    private String special;
    private String path;
    private String value;
    //省略上述属性的 get()和 set()方法
...
    //编写 execute()方法
    public String execute() throws Exception {
        //设置页面编码格式
        ServletActionContext.getResponse().setCharacterEncoding("UTF-8");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        String filePath = request.getParameter("path");    //获取路径参数
        if (function.isInvalid(title) || function.isInvalid(singer)
                                                        //判断参数是否为空
                || function.isInvalid(special) || function.isInvalid(path)){
            out.println(function.PlutoJump("任何一项都不能为空！",
            "upload.jsp?path="
                    + filePath));
        } else {
            filePath = filePath.replace("upload", "upload\\");
                                                        //获取文件后缀
            DBConnection conn = new DBConnection();     //获取数据库连接
            long time = new Date().getTime();           //获取上传时间
            //当操作数据库成功
            if (conn
                    .execute("insert into music(title,singer,special,
                    value,time,click,url) values('"
                            + title
```

```
                        + "','"
                        + singer
                        + "','"
                        + special
                        + "','"
                        + value + "','" + time + "',0,'" + filePath +
                        "')")) {
                    // 添加 TIP 信息
                    String tip = "[" + session.getAttribute("PlutoUser").
                    toString()
                            + "] 分享了歌曲 [" + title + "]";
                    conn.execute("insert into tip(value) values('" + tip +
                    "')");
                    out.println(function.PlutoJump("提交成功！", "index.jsp"));
                } else {                                    //操作数据库失败
                    out.println(function.PlutoJump("提交失败！", "upload.jsp?
                    path="
                            + filePath));
                }
            }
            return null;
        }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 Servlet 中的相关类获取所有上传的音乐信息，最后通过 SQL 语句中的 upload()方法实现音乐的更新功能。

在 struts.xml 文件中配置 upload 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 upload 的 Action-->
        <action name="upload" class="Pluto.upload"></action>
...
    </package>
</struts>
```

在线音乐管理系统中的该模块功能中涉及 upload.jsp 页面，该页面被用来作为在线音乐管理系统的更新音乐信息页面，具体内容如代码 23.26 所示。

代码 23.26　更新音乐信息页面：upload.jsp

```
...
    <body>
...
        <h2>上传音乐第一步（上传音乐）</h2>
        <form id="form1" name="form1" method="post"

    action="upload.action?path=<%=request.getParameter("path"));%>">
        <input type="hidden" name="path"value="<%=request.getParameter
        ("path"))%>" />
                <td>您的音乐已经上传成功，您可以点下面的播放器进行试听！</td>
```

```
                <td height="23" align="center">
                    <!--播放器-->
                    <object type="application/x-shockwave-flash"
                                data="player/audioplayer.swf" width="290"
                                height="24"
                                id="audioplayer7643">
                        <param name="movie" value="player/audioplayer.
                        swf" />
                        <param name="FlashVars"
    value="playerID=7643&soundFile=<%=request.getParameter("path"));%>" />
                        <param name="quality" value="high" />
                        <param name="menu" value="false" />
                        <param name="wmode" value="transparent" />
                    </object>
                </td>
                <!--歌曲名称输入框-->
                <td>
                    歌曲名称:
                    <input type="text" name="title" id="title" size="15" />
                </td>
                <td>
                    <!--歌手输入框-->
                    歌手:
                    <input type="text" name="singer" id="singer" size="15" />
                </td>
                <!--所属专辑输入框-->
                <td>
                    所属专辑:
                    <input type="text" name="special" id="special" size="15"/>
                </td>
                <!--简介输入框-->
                <td>
                    简介:
                    <input type="hidden" />
                    <textarea name="value" id="value" cols="25" rows="6">
                    </textarea>
                </td>
                <!--提交按钮-->
                <td>
                    <input type="submit" name="button" id="button" value=
                    "提交" />
                </td>
            </table>
        </form>
...
    </body>
...
```

至此，就完成了在线音乐信息更新功能。

23.4.5　实现添加评论功能

当用户登录在线音乐管理系统后，可以对其他用户上传的音乐进行评论。addComments.java 类用来实现添加评论功能，具体内容如代码 23.27 所示。

代码 23.27　实现添加评论功能：addComments.java

```
public class addComments extends ActionSupport {
    private String name;
    private String comments;
    private String id;
    //省略上述属性的 get()和 set()方法
...
    public String execute() throws IOException {
        //设置页面编码格式
        ServletActionContext.getResponse().setCharacterEncoding("UTF-8");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        if (name == null || function.isInvalid(comments)//判断是否为非法登录
                || function.isInvalid(id)) {
            out.println("非法访问！");
            return null;
        } else {
            if ("".equals(name)) {
                name = "游客";
            }
            long time = new Date().getTime();           //获取时间变量
            DBConnection conn = new DBConnection();     //获取数据库连接
            boolean insert = conn
                    .execute("insert into comments(value,name,music id,
                    time) values('"
                            + comments
                            + "','"
                            + name
                            + "',"
                            + id
                            + ",'"
                            + time
                            + "')");                    //插入结果
            if (insert) {                               //判断插入结果
                out.println("评论添加成功！");
            } else {
                out.println("评论添加失败！");
            }
        }
        return null;
    }
}
```

在 struts.xml 文件中配置 addComments 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 addcomments 的 Action-->
        <action name="addcomments" class="Pluto.addComments"></action>
```

```
...
    </package>
</struts>
```

在线音乐管理系统中的该模块功能中涉及 show.jsp 页面，该页面用来显示关于音乐评论功能，具体内容如代码 23.28 所示。

代码 23.28　音乐评论功能：show.jsp

```
...
    <body>
...
        <!--标题 -->
        <h2>我要留言</h2>
        <!--表单 -->
        <form id="myForm" name="myForm" method="post"
            action="addcomments.action?id=<%=id);%>" class="niceform">
        <!--用户名输入框 -->
        <label>昵称:
        <input type="text" name="name" id="textfield" maxlength="16" />
        (不填为游客)
        </label>
        <!--留言输入框 -->
        <label>留言:
        <input type="hidden" />
        <textarea name="comments" id="valueid" cols="20" rows="5">
        </textarea>
        </label>
        <!--提交按钮-->
        <input type="submit" name="button" id="button" value="提交" />
        </label>
        <h2 class="title">最近留言</h2>
...
    </body>
...
```

至此，就完成了添加评论的功能。

23.4.6　实现音乐盒功能

在上传完音乐后，不仅可以对上传的音乐进行评论，而且可以创建属于该用户的音乐盒。MusicBox.java 用来实现创建音乐盒的功能，具体内容如代码 23.29 所示。

代码 23.29　创建音乐盒：MusicBox.java

```
public class MusicBox extends ActionSupport {
    private String music id;                            //创建字段

    //省略上述属性的 get()和 set()方法
...
    //编写 execute 方法
    public String execute() throws IOException, SQLException {
        //设置编码格式
```

```
ServletActionContext.getResponse().setCharacterEncoding("UTF-8");
    //获取输出流对象 PrintWriter
    PrintWriter out = ServletActionContext.getResponse().getWriter();
    //设置编码格式
    ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
    ServletActionContext.getResponse().setHeader("Cache-Control",
            "no-cache");
    ServletActionContext.getResponse().setDateHeader("Expires", 0);
    //获取变量 userName
    String userName = ServletActionContext.getContext().
    getSession().get(
            "PlutoUser").toString();
    DBConnection conn = new DBConnection();        //获取数据库连接
    ResultSet rs = conn                            //查询数据库
            .executeQuery("select music box from user where name='"
                    + userName + "'");
    if (rs.next()) {                           //当数据库中不存在相同音乐
        String playList = rs.getString("music box");
        if (playList == null) {                //添加到数据库
            if (conn.execute("update user set music box='" + music id
                    + "' where name='" + userName + "' ")) {
                out.println("添加成功！");
            } else {
                out.println("出现错误！");
            }
        } else {                               //当数据库中已存在相同音乐
            String[] playListArr = playList.split(",");
            for (int i = 0; i < playListArr.length; i++) { //遍历数组
                if (music_id.equals(playListArr[i])) {
                    out.println("抱歉，您的音乐盒中已经存在此歌曲！");
                    return null;
                }
            }
            //音乐盒中是否存在其他音乐
            if (function.isInvalid(playList)) { //音乐盒中没有任何歌曲
                if (conn.execute("update user set music box='" +
                music id
                        + "' where name='" + userName + "' ")) {
                    out.println("添加成功！");
                } else {
                    out.println("出现错误！");
                }
            } else {                                //存在其他音乐
                if (conn
                    .execute("update user set music box = CONCAT(music
                    box , ','"
                            + music_id
                            + "') where name='"
                            + userName
                            + "'")) {
                    out.println("添加成功！");
                } else {
                    out.println("出现错误！");
                }
            }
        }
    } else {
        out.println("出现错误！");
```

```
            return null;
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 Servlet 的相关类获取所要的音乐信息，最后通过 SQL 语句中的 update()方法实现音乐盒功能。

在 struts.xml 文件中配置 MusicBox 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 addtobox 的 Action-->
        <action name="addtobox" class="Pluto.MusicBox"></action>
...
    </package>
</struts>
```

在数据库中每个用户都存在一个表示自己音乐盒的字段，默认为空。如果用户添加一首歌曲到自己的音乐盒中，数据库中就会存储了这首歌的 ID；若为多首歌曲则用“，”分隔开，具体内容如代码 23.30 所示。

代码 23.30　操作播放列表：setbox.java

```
public class setbox extends ActionSupport {
    //创建字段
    List<String> array = null;
    private String[] list;
    private String select;
    //省略上述属性的 get()和 set()方法
...
    //编写 execute()方法
    public String execute() throws IOException, SQLException, JDOMException {
        //设置编码格式

    ServletActionContext.getResponse().setCharacterEncoding("GB2312");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //创建 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-
        cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //获取 userName 的值
        String userName = ServletActionContext.getContext().
        getSession().get(
                "PlutoUser").toString();
        if (list == null) {                          //判断 list 的值
            out.println(function.PlutoJump("请选择歌曲！",
```

```
            "musicbox.jsp"));
        } else {
            if ("del".equals(select)) {          //删除歌曲
                DBConnection conn = new DBConnection();
                ResultSet rs = conn
                        .executeQuery("select music box from user where
                        name = '"
                                + userName + "'");
                rs.next();
                String musicList = rs.getString("music box");
                rs.close();
                if (musicList.length() == 1) { //如果其中只有一首歌，就清空
                    boolean update = conn
                            .execute("update user set music_box = NULL where
                            name = '"
                                    + userName + "'");
                    if (update) {
                        out.println(function.PlutoJump("删除成功",
                        "musicbox.jsp"));
                    } else {
                        out.println(function.PlutoJump("出现错误",
                        "musicbox.jsp"));
                    }
                } else {
                    String[] musicBoxArr = musicList.split(",");
                    if (musicBoxArr.length == list.length) {
                                          //如果是全选就直接清空
                        boolean update = conn
                                .execute("update user set music box = NULL
                                where name = '"
                                        + userName + "'");
                        if (update) {
                            out.println(function.PlutoJump("删除成功",
                                    "musicbox.jsp"));
                        } else {
                            out.println(function.PlutoJump("出现错误",
                                    "musicbox.jsp"));
                        }
                    } else {
                        array = new ArrayList<String>();
                                                  //使用链表，容易操作
                        for (int i = 0; i < musicBoxArr.length; i++) {
                            array.add(musicBoxArr[i]);
                        }
                        //寻找相同元素并从链表中删除
                        for (int i = 0; i < array.size(); i++) {
                            for (int j = 0; j < list.length; j++) {
                                if (array.get(i).toString().equals
                                (list[j])) {
                                    array.remove(i);
                                }
                            }
                        }
                        Object[] newMusicBox = array.toArray();
                                                        //再形成数组
                        String newMusic = newMusicBox[0].toString() + ",";
                                                        //形成字符串
                        for (int i = 1; i < newMusicBox.length; i++) {
                            newMusic += newMusicBox[i].toString() + ",";
```

```
                        }
                        // 去掉最后的","
                        newMusic = newMusic.substring(0, newMusic.length()
                        - 1);
                        boolean update = conn
                                .execute("update user set music box = '"
                                        + newMusic + "' where name = '"
                                        + userName + "'");
                        if (update) {
                            out.println(function.PlutoJump("删除成功",
                                    "musicbox.jsp"));
                        } else {
                            out.println(function.PlutoJump("出现错误",
                                    "musicbox.jsp"));
                        }
                    }
                }
            } else{                                     //播放歌曲
                creatXML xml = new creatXML();      //建立播放列表
                xml.bulidXML(list, request, userName);
                out.println(function.PlutoJump("建立播放列表成功! ",
                "player"));
            }
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先会从数据库中取出音乐盒中的音乐，然后利用 String[] split(String regex)方法将音乐盒中的音乐分割成为数组，但是数组对于增删改操作非常不便，所以会转存为链表进行操作。最后还需要再将其转化为数组，存储更新到数据库中。

在 struts.xml 文件中配置 setbox 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 setbox 的 Action-->
        <action name="setbox" class="Pluto.setbox"></action>
...
    </package>
</struts>
```

在线音乐管理系统中，每个用户都拥有一个 XML 文件，用来存储该用户的播放列表。creatXML.java 利用 JDom 组件动态创建 XML 播放列表，具体内容如代码 23.31 所示。

代码 23.31　创建 xml 功能：creatXML.java

```
public class creatXML {
    private String str = "3,2,4,5,6,7";
    private String [] playListArr;
    private String path;
    //省略上述属性的 set()和 get()方法
...
    public void bulidXML(String [] id,HttpServletRequest request,String
    userName) throws IOException, JDOMException, SQLException {
```

```
        //创建 XML 头
        Element playList = new Element("playlist");;
        Document Doc = new Document(playList);
        playList = Doc.getRootElement();
        playList.setAttribute("version", "1");
        playList.setAttribute("xmln","http://xspf.org/ns/0/");
        Element title = new Element("title");
        title.setText("Pluto's Player");
        playList.addContent(title);
        //头结束
        Element trackList = new Element("trackList");
        playList.addContent(trackList);
        DBConnection conn  = new DBConnection();            //获取数据库连接
        for(int i=0;i<id.length;i++){
            //获取查询结果
            ResultSet rs = conn.executeQuery("select * from music where id
            = "+id[i]+"");
            rs.next();
            String music_title = rs.getString("title");    //获取标题
            String music_singer = rs.getString("singer");  //获取歌手
            String music_url = rs.getString("url");    //获取歌曲的 URL 地址
            Element track = new Element("track");
            Element annotation = new Element("annotation");
            annotation.setText(music_title + " - " + music_singer);
            track.addContent(annotation);
            Element location = new Element("location");
            location.setText("../" + music_url);
            track.addContent(location);
            trackList.addContent(track);
        }
        //获取查询结果
        ResultSet userRs = conn.executeQuery("select id from user where name
        = '"+userName+"'");
        userRs.next();
        String user_id = userRs.getString("id");   //获取 ID
        XMLOutputter XMLOut = new XMLOutputter();  //创建 XMLOutputter 对象
        XMLOut.output(Doc, new FileOutputStream(request.getSession().
        getServletContext().getRealPath("/player/xml/" + user_id + ".xml")));
    }
}
```

【代码解析】

在上述代码中，通过 JDom 组件创建 XML 文件，而 XML 文件中各个元素的值则为数据库中音乐列表的相关值。

在线音乐管理系统中的该模块功能中涉及 Musicbox.jsp 页面，该页面被用来显示关于音乐盒的各种信息，具体内容如代码 23.32 所示。

代码 23.32　关于音乐盒页面：Musicbox.jsp

```
...
    <body>
...
        <!--标题 -->
        <h2>我的音乐盒</h2>
        <%
            if (music_box != null) {            //音乐盒中存在音乐
        %>
```

```
<!--表单 -->
<form method="post" name="form2" id="form2" action="setbox.
action">
    <!--表格头标题-->
    <TH>ID</TH>
    <TH>歌曲</TH>
    <TH>歌手</TH>
    <TH>试听</TH>
    <!--选择框-->
    <TH>
        <input type="checkbox" name="chkAll" value="" title="全选
        /取消"
                                        onclick="selectAll();" />
    </TH>
    <%
        int maxSong = limit + pagesize;    //设置变量 maxSong 的值
        //遍历结果
        for (int i = limit; i < maxSong; i++) {
        try {
            //执行 SQL 语句
            ResultSet music_rs = conn.executeQuery("select title,
            singer,url from
                music where id="+ music box arr[i] + "");
            music_rs.next();                //遍历查询结果
            String title = music rs.getString("title");
                                            //获取标题
            String singer = music rs.getString("singer");
                                            //获取歌手
            String url = music_rs.getString("url");//获取 URL 地址
            if (i % 2 == 0) {                       //判断 i 的值
                out.println("<TBODY><TR>");
            } else {
                out.println("<TBODY><TR class=odd>");
            }
            //输出关于音乐的相关信息
            out.println("<TD>" + music_box_arr[i]+ "</TD>");
            out.println("<TD>" + title + "</TD>");
            out.println("<TD>" + singer + "</TD>");
            Random rd = new Random();
            int rd id = rd.nextInt(9999);
            //关于歌曲的播放器
            String player = "<object type=application/x-shockwave-
            flash
                 data=player/audioplayer.swf width=200 height=15
                 id=audioplayer"+ rd_id + ">
                <param name=movie value=player/audioplayer.swf />
                <param name=FlashVars value=playerID="+ rd id+
                    "&soundFile="+ url+ " />
                <param name=quality value=high /><param name=menu
                    value=false />
                <param name=wmode value=transparent /></object>";
            out.println("<TD>" + player + "</TD>");
            out.println("<TD><input type=\"checkbox\" name=
            \"list\" id=\"list\"
+ music box arr[i] + "\" value=\""+ music box arr[i] + "\"></TD>");
        } catch (Exception e) {
            //数组越界就跳过
            }
```

```
                }
                out.println("</TBODY></TABLE>");
                boolean noAjax = false;
                out.println("<div class=\"yahoo2\">" function.page
                (maxPage, nowpage, pagesize,"musicbox.jsp", noAjax) +
                "</div><br />");
                out.println("<div align=center> <label>选中项：<select
                name=select id=select class=width_100> <option value=play>
                播放</option><option value=del>删除</option> </select>
                </label> <label>      <input
                type=\"submit\"  value=提交> </label>  </div></FORM>");
out.println("");
                } else {
                    out.println("对不起，暂时您的音乐盒中没有任何音乐！");}%>
...
    </body>
...
```

至此，就完成了创建音乐盒的功能。

23.4.7　实现短信发送

在线音乐管理系统中，不仅可以实现关于音乐的各种操作，而且还实现了短信的发送和接收。message.java 代码用来实现发送短信功能，具体内容如代码 23.33 所示。

代码 23.33　发送短信功能：message.java

```
public class message extends ActionSupport {
    private String to;
    private String value;
    private String title;
    //省略上述属性的 get()和 set()方法
...
    public String execute() throws Exception {
        //设置编码格式
        ServletActionContext.getResponse().setCharacterEncoding("GB2312");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //设置页面的相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //获取 userName 的值
        String userName = ServletActionContext.getContext().getSession().
        get(
                "PlutoUser").toString();
        if (function.isInvalid(to)) {                      //判断 to
            out.println(function.PlutoJump("请填入收件人名称！", "message.
            jsp"));
        }
        if (function.isInvalid(value)) {                   //判断 value
            out.println(function.PlutoJump("请填入消息内容！", "message.
            jsp"));
        }
```

```
        DBConnection conn = new DBConnection();       //获取数据库连接
        //获取执行结果
        ResultSet rs = conn
                .executeQuery("select id,name from user where name='" + to
                        + "'");
        //遍历结果
        if (rs.next()) {
            String to_name = rs.getString("name");    //获取用户名变量
            if (userName.equals(to_name)) {
                out
                        .println(function.PlutoJump("对不起，不能给自己发短
                        信息！",
                                "message.jsp"));
            } else {
                long time = new Date().getTime();      //获取时间变量
                int to_id = rs.getInt("id");            //获取 ID 变量
                if (conn
                        .execute("insert into message('from','to',title,
                        value,time,new) values('"
                                + userName
                                + "','"
                                + to_id
                                + "','"
                                + title
                                + "','" + value + "','" + time + "',1)")) {
                    out.println(function.PlutoJump("发送短消息成功！",
                    "message.jsp"));
                } else {
                    out.println(function.PlutoJump("发送短消息失败！",
                    "message.jsp"));
                }
            }
        } else {
            out.println(function.PlutoJump("用户不存在！", "message.jsp"));
        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 Servlet 的相关类获取所要发送短信的相关信息，最后通过 SQL 语句中的 select()方法来查找所要发送的对象并利用 insert()方法实现发送功能。

在 struts.xml 文件中配置 message 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 message 的 Action-->
        <action name="message" class="Pluto.message"></action>
...
    </package>
</struts>
```

在线音乐管理系统中的该模块功能中涉及 message.jsp 页面，该页面用来实现短信的发

送，具体内容如代码 23.34 所示。

代码 23.34　短信发送页面：message.jsp

```
...
    <body>
...
        <!--标题-->
        <h2>发送短消息! </h2>
        <!--表单-->
        <form id="form1" name="form1" method="post"
            action="message.action" class="niceform">
        <!--收件人输入框-->
        <label>收件人:
        <input type="text" name="to" id="textfield" maxlength="16" />
        </label>
        <!--标题输入框-->
        <label>标 题:
        <input type="text" name="title" id="textfield" maxlength="16" />
        </label>
        <!--内容输入框-->
        <label>内容:
        <input type="hidden" />
        <textarea name="value" id="textarea" cols="30" rows="5">
        </textarea>
        </label>
        <label>
        <!--提交按钮-->
        <input type="submit" name="button" id="button" value="提交" />
        </label>
        </form>
...
    </body>
...
```

至此，就完成了发送短信的功能。

23.4.8　实现短信删除

在线音乐管理系统中，不仅可以实现关于音乐的各种操作，而且还实现了短信的发送和删除功能。delmessage.java 代码用来实现删除短信功能，具体内容如代码 23.35 所示。

代码 23.35　删除短信功能：delmessage.java

```
public class delmessage extends ActionSupport {
    private String[] list;
    //省略上述属性的 get()和 set()方法
...
    //编写 execute()方法
    public String execute() throws IOException, SQLException {
        //设置编码格式
        ServletActionContext.getResponse().setCharacterEncoding("GB2312");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
```

```
        //获取 request 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        //设置页面一些相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //获取 userName 的值
        String userName = ServletActionContext.getContext().getSession().
        get(
                "PlutoUser").toString();
        if (list == null) {
            out.println(function.PlutoJump("请选择短消息！", "message.
            jsp"));
        } else {
            DBConnection conn = new DBConnection();        //获取数据库连接
            if (list.length == 1) {
                //删除操作的结果
                boolean del = conn.execute("delete from message where
                id="+list[0]+" limit 1");
                if(del){
                    out.println(function.PlutoJump("删除成功！", "message.
                    jsp"));
                }else{
                    out.println(function.PlutoJump("删除失败！", "message.
                    jsp"));
                }
            }else {
                String sql = "delete from message where id=" + list[0];
                for(int i=1;i<list.length;i++){
                    sql += " or id=" + list[i];
                }
                boolean del = conn.execute(sql);
                if(del){                                   //判断操作
                    out.println(function.PlutoJump("删除成功！", "message.
                    jsp"));
                }else{
                    out.println(function.PlutoJump("删除失败！", "message.
                    jsp"));
                }
            }

        }
        return null;
    }
}
```

【代码解析】

在上述代码中，首先通过 Struts 2.0 中的相关类来获取 Servlet 的相关类，然后通过 Servlet 的相关类获取所要删除短信的相应信息，最后通过 SQL 语句中的 delete()方法把短信的相应信息删除。

在 struts.xml 文件中配置 delmessage 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
```

```
        <!-- 配置名为 delmessage 的 Action-->
        <action name="delmessage" class="Pluto.delmessage"></action>
...
    </package>
</struts>
```

在线音乐管理系统中的该模块功能中涉及 showmessage.jsp 页面，该页面用来实现短信的接收，具体内容如代码 23.36 所示。

代码 23.36　短信接收页面：showmessage.jsp

```
...
    <body>
        <%
            String message_id = request.getParameter("id"); //获取 ID 参数
                //执行相应的查询语句
                ResultSet rs = conn
                        .executeQuery("select title,value from message
                        where id="
                                + message id + "");
                //遍历执行结果
                rs.next();
                String title = rs.getString("title");      //获取标题
                String value = rs.getString("value");      //获取值
                //执行更新语句
                conn.execute("update message set new=0 where id=" + message
                id
                        + "");
        %>
        <TABLE width="80%" align="center" class="mytable">
                    <td class=odd>
                        <!--获取接受到短信的标题-->
                        <th scope=col> <%=title);%></th>
                    </td>
                    <!--获取接受到短信的值-->
                    <td><%=value)%></td>
        </TABLE>
    </body>
...
```

至此，就完成了接收短信的功能。

23.4.9　实现点歌功能

在线音乐管理系统中，不仅可以实现短信的发送和接收，而且还可以实现音乐的发送和接收。sendmusic.java 代码用来实现点歌功能，具体内容如代码 23.37 所示。

代码 23.37　点歌功能：sendmusic.java

```
public class sendmusic extends ActionSupport {
    private String to;
    private String value;
    private String hidename;
    private String id;
    //省略上述属性的 get()和 set()方法
...
```

```
    public String execute() throws Exception {
        //设置编码格式
        ServletActionContext.getResponse().setCharacterEncoding("GB2312");
        //获取输出流
        PrintWriter out = ServletActionContext.getResponse().getWriter();
        //获取 HttpServletRequest 对象
        HttpServletRequest request = ServletActionContext.getRequest();
        //设置相关信息
        ServletActionContext.getResponse().setHeader("Pragma", "No-cache");
        ServletActionContext.getResponse().setHeader("Cache-Control",
                "no-cache");
        ServletActionContext.getResponse().setDateHeader("Expires", 0);
        //获取用户名
        String userName = ServletActionContext.getContext().getSession().
        get(
                "PlutoUser").toString();
        if (function.isInvalid(to)) {                      //校验变量 to
            out.println(function.PlutoJump("你要发给谁呢？？", "index.
            jsp"));
        }
        if (function.isInvalid(value)) {                   //校验变量 value
            value = "这家伙很懒，什么都没有留下！";
        }
        DBConnection conn = new DBConnection();
                                                  //获取数据库连接 DBConnection
        ResultSet to rs = conn.executeQuery("select id,name from user where
        name='"
                + to + "'");                               //获取相应的执行结果
        if (to_rs.next()) {                                //遍历结果
            String to_id = to_rs.getString("id");          //获取 ID 号
            String to_name = to_rs.getString("name");      //获取名称
            if(to_name.equals(userName)){                  //判断 to_name 变量
                out.println(function.PlutoJump("不能给自己点歌！", "index.
                jsp"));
            }
            if ("true".equals(hidename)) {                 //判断是否是匿名点歌
                userName = "匿名";
            }
            ResultSet music_rs = conn                      //获取查询结果
                    .executeQuery("select title,url from music where id="
                    + id+ "");
            music rs.next();                               //遍历结果
            //获取音乐的各个参数
            int rd id = new Random().nextInt(9999);
            String music_title = music_rs.getString("title");
            String music url = music rs.getString("url");
            music_url = music_url.replace("upload\\", "upload\\\\");
            String title = "[" + userName + "]为您点播了一首[" + music_title
            + "]";
            String message value = "<p>他（她）给您的留言：" + value + "</p>";
            //设置变量 message_value 的值
            message_value += "<p>您可以点击下面的播放器进行试听！<br />";
            message_value += "<object type=\"application/x-shockwave-
            flash\" data=\"player/audioplayer.swf\" width=\"290\" height=
            \"24\"  id=\"audioplayer"
                    + rd id
                    + "\">  <param name=\"movie\" value=\"player/
```

```
                audioplayer.swf\" /><param name=\"FlashVars\"
                value=\"playerID="
                + rd_id
                + "&soundFile="
                + music_url
                + "\" />    <param name=\"quality\" value=\"high\" />
                <param name=\"menu\" value=\"false\" /><param name=
                \"wmode\" value=\"transparent\" /></object><br />";
        long time = new Date().getTime();
        //设置插入语句
        boolean insert = conn
                .execute("insert into message('from','to','title',
                'value','time','new') values('"
                        + userName
                        + "','"
                        + to_id
                        + "','"
                        + title
                        + "','"
                        + message_value + "','" + time + "',1)");
        if (insert) {                                    //判断操作
            out.println(function.PlutoJump("点播成功！", "index.jsp"));
        } else {
            out.println(function.PlutoJump("点播失败！", "index.jsp"));
        }
    } else {
        out.println(function.PlutoJump("你要发给谁呢？？", "index.
        jsp"));
    }
    return null;
  }
}
```

注意：上述代码与 23.4.7 节中实现“短信发送功能”功能的 message.java 代码基本相同。

在 struts.xml 文件中配置 sendmusic 类。

```
<struts>
    <constant name="struts.multipart.maxSize" value="10000000" />
    <package name="Pluto" extends="struts-default">
...
        <!-- 配置名为 sendmusic 的 Action-->
        <action name="sendmusic" class="Pluto.sendmusic"></action>
...
    </package>
</struts>
```

在线音乐管理系统中的该模块功能中涉及 sendmusic.jsp 页面，该页面用来实现点歌的功能，具体内容如代码 23.38 所示。

代码 23.38　关于点歌页面：sendmusic.jsp

```
...
    <BODY>
        <form action="sendmusic.action?id=<%=id)%>" method="post">
            <!--标题-->
            <TD>点播歌曲</TD>
...
```

```
            <!--用户名输入框-->
            <TD>用户名:
            <INPUT type="text" name="to" maxlength="16" />
            </TD>
            <!--匿名点歌选择框-->
            <TD>匿名点歌
            <input type="checkbox" name="hidename" value="true"  id=
            "checkbox">
            </TD>
            <!--留言输入框-->
            <TD>留言:
            <textarea name="value" id="textarea" cols="45" rows="5">
            </textarea>
            </TD>
            <TD>
            <!--确定和取消按钮-->
            <INPUT id=Login type=submit value="确 定">
            <INPUT type="reset" onclick=tb_remove(); value="取 消" />
            </TD>
        </form>
    </BODY>
...
```

至此，就完成了点歌的功能。

23.5　小　　结

本章主要介绍一个完整的在线音乐管理系统，该系统基于 AJAX+JSP+Struts 2.x 框架构建而成。在具体实现在线音乐管理系统时，根据不同用户分成两部分：关于超级管理员的操作和关于注册用户的操作。该系统不仅实现了注册、登录和修改超级管理员密码、删除注册用户、删除上传音乐，以及操作友情链接等关于超级管理员的操作功能，而且还实现了用户注册、登录、退出功能，实现了音乐的上传和音乐信息更新、添加用户评论功能，实现了创建音乐盒、短信的发送和接收、歌曲的点歌等关于注册用户的操作功能。

第 24 章　数据汇聚系统（Struts 2.x+Spring+iBATIS）

对于一个大型网络系统，在具体开发时数据源不是由一个地方产生，而是由好多不同地方的数据源汇聚而成，所以能够实现数据自动获取、数据自动更新的数据汇聚系统是一个必不可少的模块。

本章将通过 Struts 2.x+Spring+iBATIS 框架技术来实现一个数据汇聚系统，其中持久层使用 iBATIS 框架，数据库管理系统则为微软公司的 SQL Service 2000。

24.1　数据汇聚系统简述

为了让读者可以快速地理解和掌握数据汇聚系统，在具体讲解时把该系统按照 Java EE 开发标准的 4 层结构体系分成 4 层：领域模型层、业务层、持久层和表示层。Struts 2.x 框架用来实现控制页面跳转，Spring 框架用来管理 Struts 2.x 框架中的 Action，iBATIS 框架则用来实现持久层和控制事务。该系统的功能分析如图 24.1 所示。

24.1.1　数据汇聚系统概述

对于数据分析者、决策者来说，能够获取具有有效性、准确性和及时性的数据是非常重要的，而对于这种数据的汇聚系统依靠现有的技术来实现并不难。例如对于以前依靠下属部门的层层上报数据方式已经不能满足对及时性的需求，所以该系统需要利用网络传输实现数据的传输。

为了讲清楚数据汇聚系统，下面通过一个具体的业务来讲解该功能。

总部在美国的老板每天需要分析从日本和英国两个分公司传送过来的销售业绩。为了能够实现该功能，日本分公司必须在 21 点之前把产品的销售数据存储到日本本地数据库，同时英国分公司也必须在 21 点之前把产品的销售数据存储到英国本地数据库，而美国总部当地的服务器必须在凌晨 7 点触发数据汇聚系统，自动收集英国和日本两地的数据库数据存储到本地的数据库中供老板分析、决策。

通过上述的具体业务可以知道，数据汇聚系统分为 5 个部分，分别为：

- 在美国服务器端设定数据汇聚系统；
- 数据汇聚器采集数据源数据库数据（英国）；
- 数据汇聚器采集数据源数据库数据（日本）；
- 数据汇聚器将数据存储到本地数据库（美国）；

❑ 浏览者（美国）登录后可以浏览本地数据库中存储的数据。

如何设计数据汇聚系统呢？结合 Struts 2.x+Spring+iBATIS 框架，可以通过如图 24.2 所示的流程来实现数据汇聚系统。

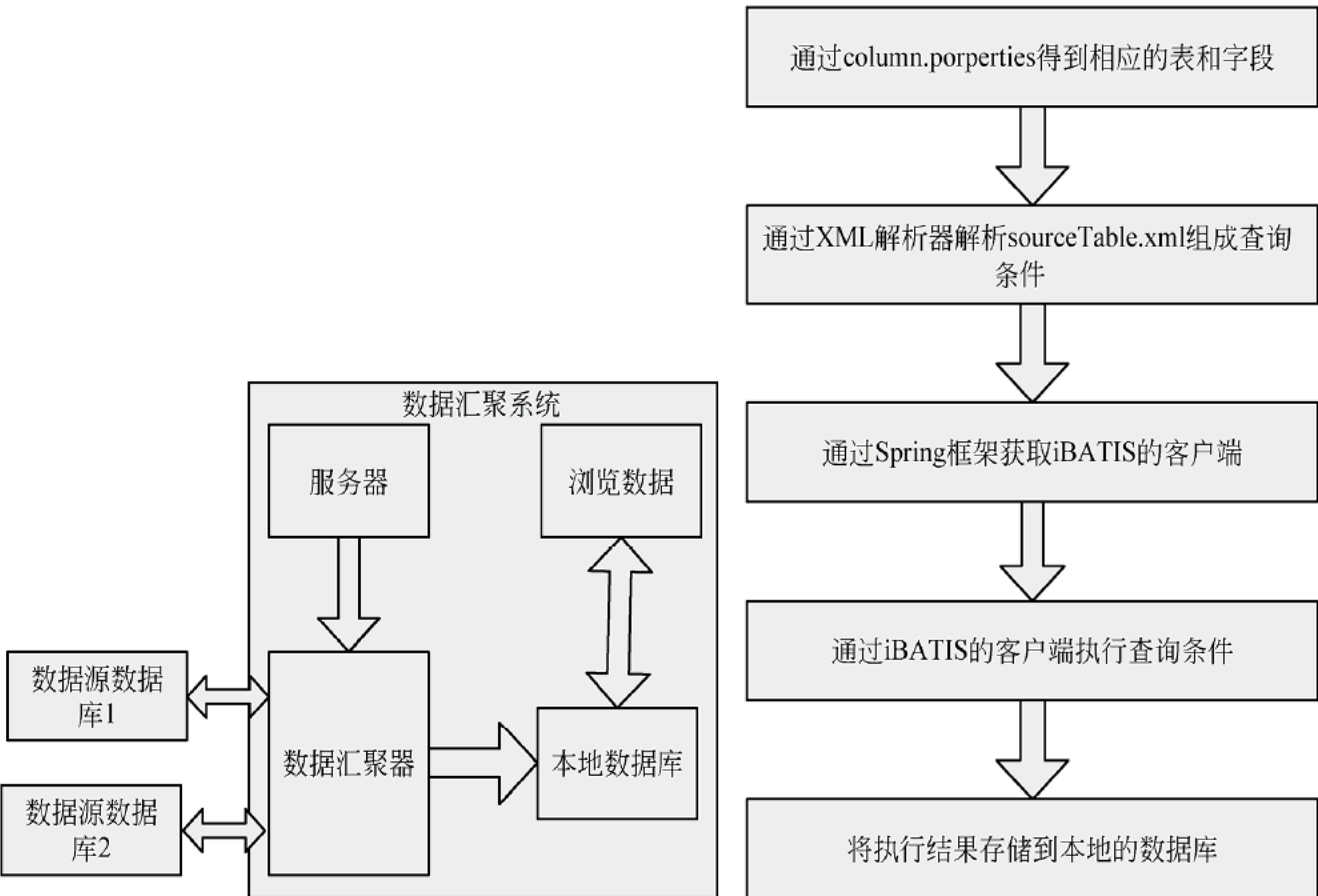


图 24.1　数据汇聚系统功能　　　　图 24.2　具体流程

24.1.2　数据汇聚系统描述

在本节中，将以直观的方式向读者介绍整个数据汇聚系统要实现的功能。这些功能包括用户登录、查看表 site 数据和查看表 bss 数据。

1．用户登录功能

用户可以通过浏览 jsp 文件夹下的 login.jsp 页面打开实现用户登录页面（如图 24.3 所示），在该页面填写相应信息以实现用户登录功能。当用户名和密码正确时，则转到系统的主界面——欢迎页面（如图 24.4 所示），否则就会转到如图 24.5 所示的登录失败页面。



图 24.3　登录页面

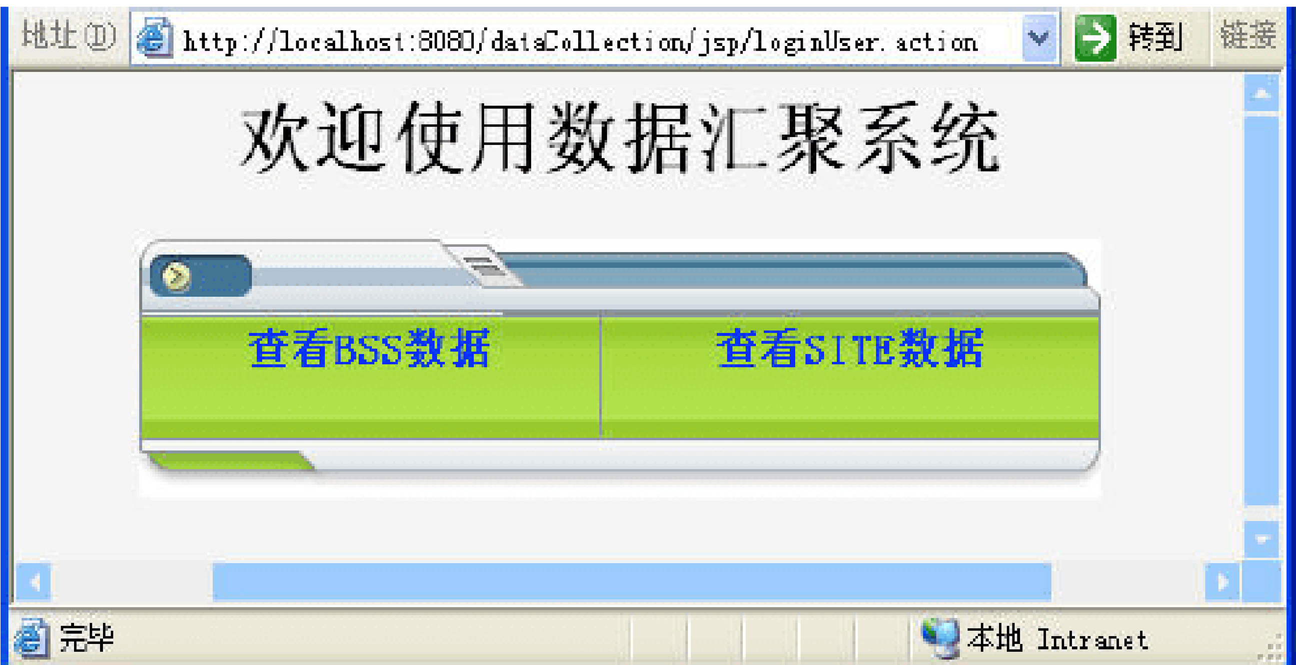


图 24.4　系统主界面

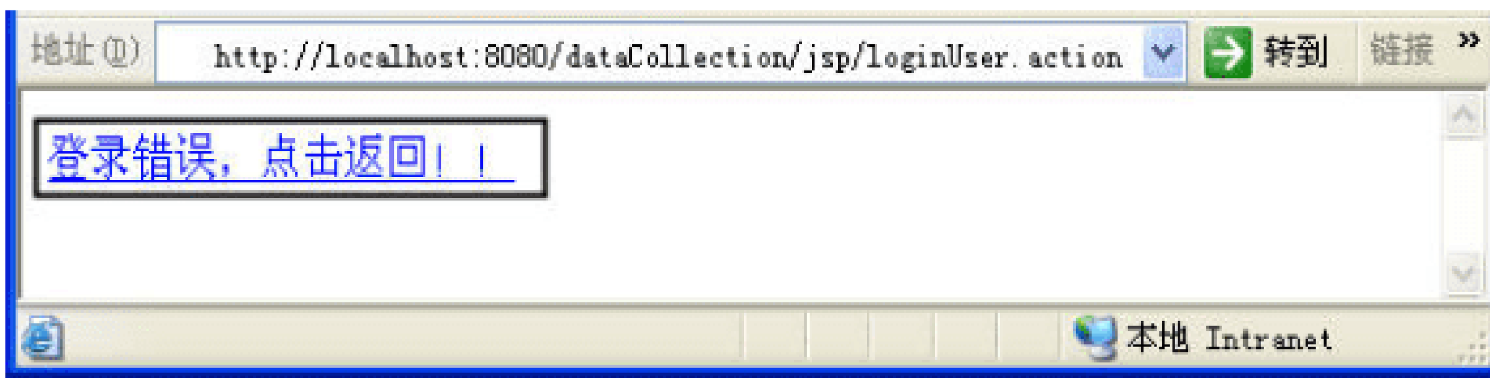


图 24.5　用户登录失败页面

2．查看site表数据

在欢迎页面中单击“查看 SITE 数据”链接（如图 24.6）就可以打开显示该表所有记录的页面，如图 24.7 所示。

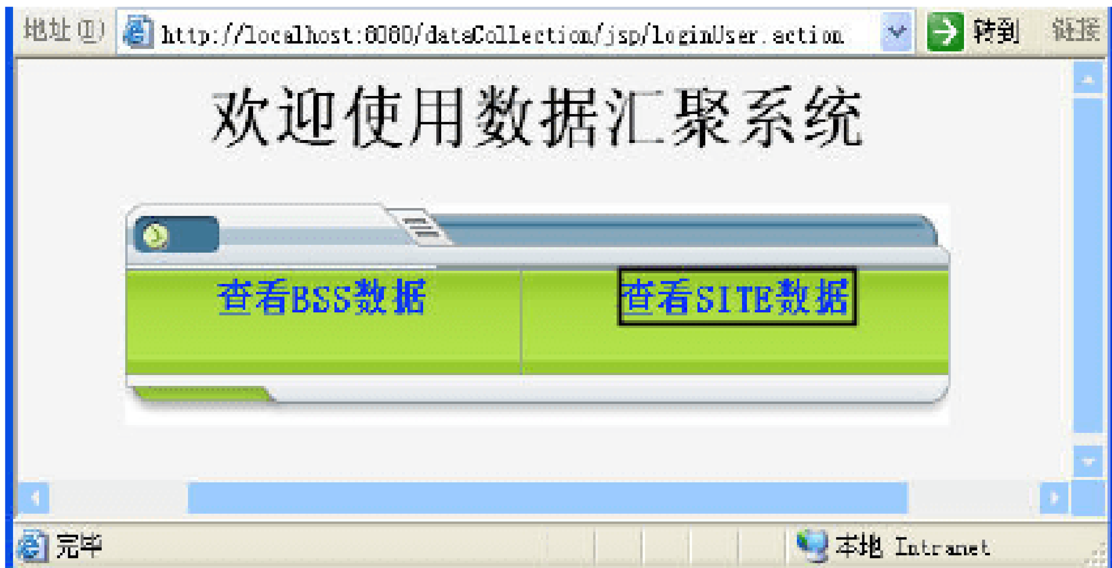




图 24.6　单击查看 SITE 数据　　　　　图 24.7　显示表格 site 的数据

3．查看bss表数据

在欢迎页面中单击“查看 BSS 数据”链接（如图 24.8）就可以打开显示该表格所有记录的页面，如图 24.9 所示。

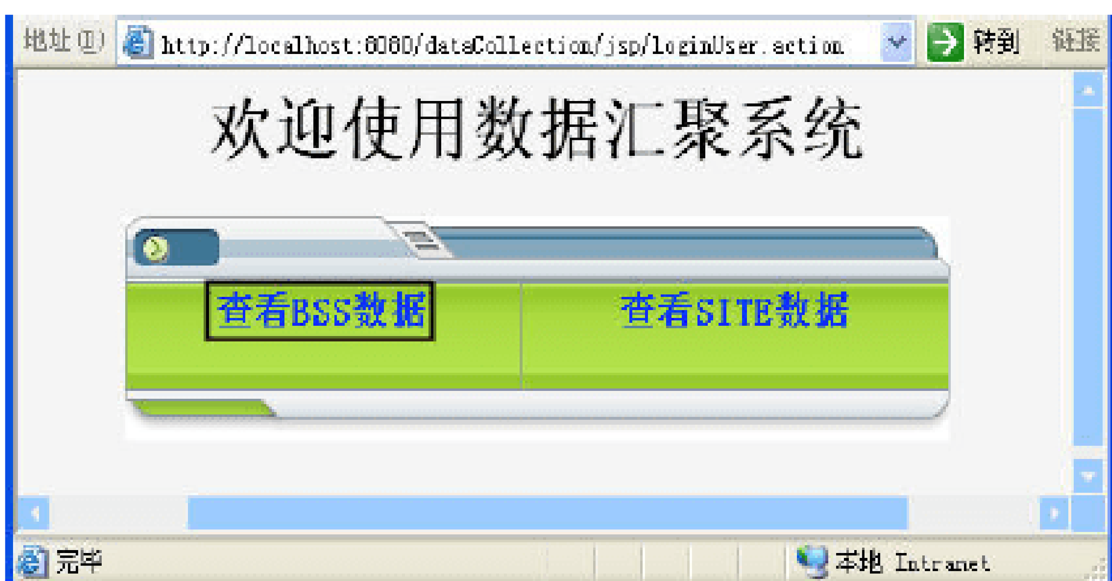




图 24.8　单击查看 BSS 数据　　　　　图 24.9　查看 bss 数据详情

24.2　数据汇聚系统简述

为了让读者可以快速地理解和掌握数据汇聚系统，在具体讲解时先按照面向应用的方式对该系统进行分层，然后再对各个应用进行 MVC 分层。在整个数据汇聚系统采用的框架中，Struts 2.x 框架用来控制页面跳转，Spring 框架用来管理 Struts 2.x 框架中的 Action，iBATIS 框架则用来实现持久层和控制事务。

24.2.1　设计数据库——远程数据库

数据汇聚系统的远程数据库管理系统中需要建立一个数据库，并在该数据库中建立两张表，存放表的数据库 Englishdata、存放数据的表 bssdata 与 sitedata。

1．创建数据库Englishdata

如果想创建出数据库 Englishdata，可以在 SQL Server 2000 的查询分析器窗口中输入如下命令：

```
CREATE DATABASE 'Englishdata'
```

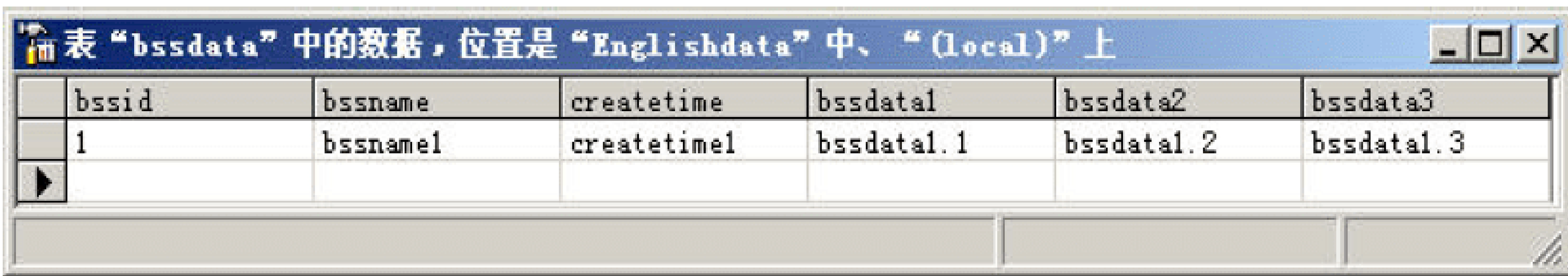
2．创建表bssdata

如果想创建出表 bssdata，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表的具体信息如表 24.1 所示。

表 24.1　表bssdata信息

字 段 名 称	数 据 类 型	字 段 说 明
bssid	int	bbs 数据库 ID 标识
bssname	varchar(50)	bbs 数据库信息名称
createtime	varchar(50)	数据创建时间
bssdata1	varchar(50)	数据1
bssdata2	varchar(50)	数据2
bssdata3	varchar(50)	数据3

为了便于测试，该表格的模拟数据如图 24.10 所示。



表“bssdata”中的数据，位置是“Englishdata”中、“(local)”上

bssid	bssname	createtime	bssdata1	bssdata2	bssdata3
1	bssname1	createtime1	bssdata1.1	bssdata1.2	bssdata1.3

图 24.10　数据库 Englishdata 中的表格 bssdata

3．创建表格sitedata

如果想创建出表 sitedata，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表的具体信息如表 24.2 所示。

表 24.2　表sitedata信息

字 段 名 称	数 据 类 型	字 段 说 明
siteid	int	bbs 数据库 ID 标识
sitename	varchar(50)	bbs 数据库信息名称
sitetime	varchar(50)	数据创建时间
sitedata1	varchar(50)	数据1
sitedata2	varchar(50)	数据2

为了便于测试，该表的模拟数据如图 24.11 所示。

表“sitedata”中的数据，位置是“Englishdata”中、“(local)”上

siteid	sitename	createtime	sitedata1	sitedata2
2	sitename1	createtime1	sitename1.1	sitename1.2

图 24.11　数据库 Englishdata 中的表格 sitedata

由于两个远程数据库 Japandata 与 Englishdata 都用来存储数据，所以这两个数据库中的表字段完全相同。最后，对于 Japandata 数据库中 bssdata 表的模拟数据如图 24.12 所示，而 sitedata 表的模拟数据如图 24.13 所示。

表“bssdata”中的数据，位置是“Japandata”中、“(local)”上

bssid	bssname	createtime	bssdata1	bssdata2	bssdata3
1	bssname2	createtime2	bssdata2.1	bssdata2.2	bssdata2.3

图 24.12　数据库 Japandata 中的表格 bssdata

表“sitedata”中的数据，位置是“Japandata”中、“(local)”上

siteid	sitename	createtime	sitedata1	sitedata2
2	sitename2	createtime2	sitedata2.1	sitedata2.2

图 24.13　数据库 Japandata 中的表格 sitedata

24.2.2　设计数据库——本地数据库

本地数据库由于是由美国总部来查看，所以命名为 Americaldata，主要用来存储采集到的资源与管理员信息，因此需要一共设计 3 张表格：bssdata、sitedata 和 userinfo。

1．创建数据库Americaldata

如果想创建出数据库 Americaldata，可以在 SQL Server 2000 的查询分析器窗口输入如下命令：

```
CREATE DATABASE 'Americaldata'
```

2．创建表格bssdata

如果想创建出表 bssdata，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表的具体信息如表 24.3 所示。

表 24.3　表bssdata信息

字 段 名 称	数 据 类 型	字 段 说 明
bssid	Int	bbs 数据库 ID 标识
bssname	varchar(50)	bbs 数据库信息名称
createtime	varchar(50)	数据创建时间
bssdata1	varchar(50)	数据1
bssdata2	varchar(50)	数据2
bssdata3	varchar(50)	数据3
proid	varchar(50)	省份 ID 号
proname	varchar(50)	省份名称

3. 创建表格sitedata

如果想创建出表 sitedata，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表的具体信息如表 24.4 所示。

表 24.4　表sitedata信息

字 段 名 称	数 据 类 型	字 段 说 明
siteid	int	Bbs 数据库 ID 标识
sitename	varchar(50)	bbs 数据库信息名称
sitetime	varchar(50)	数据创建时间
sitedata1	varchar(50)	数据1
sitedata2	varchar(50)	数据2
proid	varchar(50)	省份 ID 号
proname	varchar(50)	省份名称

由于美国本地数据库跟英国和日本数据库中都存在 bssdata 和 sitedata 表格，并且这些表的结构基本相同，所以可以通过一个名叫 ParameterObject 的类来实现持久化，该类的具体内容如代码 24.1 所示。

代码 24.1　用户模型：ParameterObject.java

```
...
public class ParameterObject {
   private String createtime;                //创建属性 createtime
   private String bssname;                   //创建属性 bssname
   private int bssid;                        //创建属性 bssid
   private String sitename;                  //创建属性 sitename
   private int siteid;                       //创建属性 siteid
   private String proid;                     //创建属性 proid
   private String proname;                   //创建属性 proname
   private String bssdata1;                  //创建属性 bssdata1
   private String bssdata2;                  //创建属性 bssdata2
   private String bssdata3;                  //创建属性 bssdata3
   private String sitedata1;                 //创建属性 sitedata1
   private String sitedata2;                 //创建属性 sitedata2
   //省略上述字段的 get()和 set()方法
...
}
```

4．创建表userinfo

如果想创建出表 userinfo，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表的具体信息如表 24.5 所示。

表 24.5　表sitedata信息

字 段 名 称	数 据 类 型	字 段 说 明
id	int	管理员编号
username	varchar(50)	管理员姓名
password	varchar(50)	管理员密码

为了便于测试，该表格的模拟数据如图 24.14 所示。



id	username	password
1	cjgong	123456

图 24.14　数据库 Americaldata 中的表格 userinfo

实现用户模型的持久化类的具体内容如代码 24.2 所示。

代码 24.2　用户模型：UserInfoVo.java

```
...
public class UserInfoVo {
    private int uiId;                                           //创建 uiId 属性
    private String username;                                    //创建 username 属性
private String password;                                        //创建 password 属性
    //省略上述属性的 get()和 set()方法
...
}
```

24.3　关于 iBATIS 框架的一些文件

当 iBATIS 框架要实现数据库方面的交互时，必须要实现关于数据库方面的配置文件和对持久化类的映射文件，本节将详细讲解这些配置文件。

24.3.1　关于持久化类（ParameterObject 和 UserInfoVo）映射文件

在数据汇聚系统中，基于数据库表映射持久化类 ParameterObject.java 和 UserInfoVo.java 的映射文件为 localData.xml，该文件的具体内容如代码 24.3 所示。

代码 24.3　持久化类映射文件：localData.xml

```
...
<!--利用 Ibatis 占位符获取动态的 SQL 语句向本地数据库插入目标数据库信息-->
    <insert id="insertData">$sqlstr$</insert>
    <!--利用 resultMap 配置数据库表列与 pojo 类属性映射-->
    <resultMap id="select bssdata"
        class="com.cjg.utils.ParameterObject">
        <result property="bssid" column="bssid" />
        <result property="bssname" column="bssname" />
        <result property="createtime" column="createtime" />
        <result property="bssdata1" column="bssdata1" />
        <result property="bssdata2" column="bssdata2" />
        <result property="bssdata3" column="bssdata3" />
        <result property="proid" column="proid" />
        <result property="proname" column="proname" />
    </resultMap>
    <!--利用 resultMap 配置数据库表列与 pojo 类属性映射-->
    <resultMap id="select sitedata"
        class="com.cjg.utils.ParameterObject">
        <result property="siteid" column="siteid" />
        <result property="sitename" column="sitename" />
        <result property="createtime" column="createtime" />
        <result property="sitedata1" column="sitedata1" />
        <result property="sitedata2" column="sitedata2" />
        <result property="proid" column="proid" />
        <result property="proname" column="proname" />
    </resultMap>
    <!--遍历本地数据库表 bssdata-->
    <select id="selectBssData" resultMap="select bssdata">
        select
        bssid,bssname,createtime,bssdata1,bssdata2,bssdata3,proid,pron-
        ame
        from bssdata
    </select>
    <!--遍历 sitedata 数据库-->
    <select id="selectSiteData" resultMap="select_sitedata">
        select
        siteid,sitename,createtime,sitedata1,sitedata2,proid,proname
        from sitedata
    </select>
    <!--配置管理员用户的数据库表列与 pojo 类映射-->
    <resultMap id="select userinfo"
        class="com.cjg.local.domain.UserInfoVo">
        <result property="username" column="username" />
        <result property="password" column="password" />
    </resultMap>
    <!--根据用户输入的用户名遍历 userinfo 表获取其对应的密码-->
    <select id="loginUser" resultMap="select userinfo">
       select * from userinfo where username=#username#
    </select>
</sqlMap>
```

【代码解析】

在上述代码中不仅实现了关于数据库表格（bssdata、sitedata）与持久层类 ParameterObject、数据库表格（userinfo）与持久层类 UserInfoVo 的映射文件，而且还设计了关于数据库表操作的具体 SQL 语句。

24.3.2　关于 Englishdata 数据库配置文件

在数据汇聚系统中基于数据库 Englishdata 的配置文件为 sourceData1.xml，该文件的具体内容如代码 24.4 所示。

代码 24.4　Englishdata 配置文件：sourceData1.xml

```
...
<sqlMapConfig>
   <!--配置 Ibatis 的各种属性信息-->
   <settings
      cacheModelsEnabled="true"
      enhancementEnabled="true"
      lazyLoadingEnabled="true"
      errorTracingEnabled="true"
      maxRequests="32"
      maxSessions="10"
      maxTransactions="5"
      useStatementNamespaces="false"
      />
    <!--配置 Ibatis 事务类型为 JDBC 与数据库信息-->
   <transactionManager type="JDBC">
      <dataSource type="SIMPLE">
         <!--配置数据库 JDBC-->
         <property name="JDBC.Driver"
            value="com.microsoft.jdbc.sqlserver.SQLServerDriver"/>
         <!--配置数据库连接地址-->
         <property name="JDBC.ConnectionURL"
            value="jdbc:microsoft:sqlserver://localhost:1433;Database-
            Name=Englishdata"/>
         <!--配置用户名-->
         <property name="JDBC.Username" value="sa"/>
         <!--配置密码-->
         <property name="JDBC.Password" value="root"/>
         <property name="Pool.MaximumActiveConnections"
            value="10"/>
         <property name="Pool.MaximumIdleConnections" value="5"/>
         <property name="Pool.MaximumCheckoutTime"
            value="120000"/>
         <property name="Pool.TimeToWait" value="500"/>
         <property name="Pool.PingQuery" value="select 1 from
         userinfo"/>
         <property name="Pool.PingEnabled" value="false"/>
         <property name="Pool.PingConnectionsOlderThan"
            value="1"/>
         <property name="Pool.PingConnectionsNotUsedFor"
            value="1"/>
      </dataSource>
   </transactionManager>
   <!--配置映射文件-->
   <sqlMap resource="com/cjg/remote/persistence/sqlmap/remoteData.xml"/>
</sqlMapConfig>
```

注意：在上述代码中配置了 iBATIS 关于数据库 Englishdata 的各项信息，即事物、数据源、连接池和映射文件路径。

24.3.3　关于 Japandata 数据库配置文件

在数据汇聚系统中基于数据库 Japandata 的配置文件为 sourceData2.xml，该文件的具体内容如代码 24.5 所示。

代码 24. 5　Japandata 配置文件：sourceData2.xml

```
...
<sqlMapConfig>
    <!--配置 Ibatis 的各种属性信息-->
    <settings
        cacheModelsEnabled="true"
        enhancementEnabled="true"
        lazyLoadingEnabled="true"
        errorTracingEnabled="true"
        maxRequests="32"
        maxSessions="10"
        maxTransactions="5"
        useStatementNamespaces="false"
        />
        <!--配置 Ibatis 事务类型为 JDBC 与数据库信息-->
    <transactionManager type="JDBC">
        <dataSource type="SIMPLE">
            <!--配置数据库 JDBC-->
            <property name="JDBC.Driver"
                value="com.microsoft.jdbc.sqlserver.SQLServerDriver"/>
            <!--配置数据库连接地址-->
            <property name="JDBC.ConnectionURL"
                value="jdbc:microsoft:sqlserver://localhost:1433;Database-
                Name=Englishdata"/>
            <!--配置用户名-->
            <property name="JDBC.Username" value="sa"/>
            <!--配置密码-->
            <property name="JDBC.Password" value="root"/>
            <property name="Pool.MaximumActiveConnections"
                value="10"/>
            <property name="Pool.MaximumIdleConnections" value="5"/>
            <property name="Pool.MaximumCheckoutTime"
                value="120000"/>
            <property name="Pool.TimeToWait" value="500"/>
            <property name="Pool.PingQuery" value="select 1 from
            userinfo"/>
            <property name="Pool.PingEnabled" value="false"/>
            <property name="Pool.PingConnectionsOlderThan"
                value="1"/>
            <property name="Pool.PingConnectionsNotUsedFor"
                value="1"/>
        </dataSource>
    </transactionManager>
     <!--配置映射文件-->
    <sqlMap resource="com/cjg/remote/persistence/sqlmap/remoteData.xml"/>
</sqlMapConfig>
```

注意：在上述代码中配置了 iBATIS 关于数据库 Japandata 的各项信息，即事物、数据源、连接池和映射文件路径。

24.3.4　关于 Americaldata 数据库配置文件

在数据汇聚系统中基于数据库 Americaldata 的配置文件共有 3 个，分别为 sql-map-config.xml、column.properties 和 sourceTable.xml。本节将详细介绍关于 Americaldata 的各个配置文件。

1．sql-map-config.xml配置文件

sql-map-config.xml 文件为配置 iBATIS 框架应用到本地的映射文件，该文件的具体内容如代码 24.6 所示。

代码 24.6　配置 Americaldata 信息：sql-map-config.xml

```
...
<sqlMapConfig>
   <settings
      cacheModelsEnabled="true"
      enhancementEnabled="true"
      maxSessions="64"
      maxTransactions="8"
      maxRequests="128"/>
      <!--配置映射文件-->
   <sqlMap resource="com/cjg/local/persistence/sqlmap/localData.xml"/>
</sqlMapConfig>
```

注意：在上述代码中主要用来配置数据库 Americaldata 的信息。

2．column.properties属性文件

column.properties 文件用来存储关于数据库 Americaldata 表的名称和字段信息，该文件的具体内容如代码 24.7 所示。

代码 24.7　存储数据库信息：column.properties

```
#关于 BSSDATA 数据库的信息
BSSDATA=(bssid,bssname,createtime,bssdata1,bssdata2,bssdata3,proid,pron
ame)
#关于 SITEDATA 数据库的信息
SITEDATA=(siteid,sitename,createtime,sitedata1,sitedata2,proid,proname)
```

读取 column 属性文件的工具类为 ParameterColumn.java，该文件的具体内容如代码 24.8 所示。

代码 24.8　读取 column 属性文件：ParameterColumn.java

```
...
public class ParameterColumn {
    //获得资源表字段与 SQL 语句
    public Map getColumnContent(String path) throws IOException {
        ClassLoader classLoader = Thread.currentThread()
                .getContextClassLoader();          //创建 classLoader 变量
```

```
        //获取输入流
        InputStream inputStream = classLoader.getResourceAsStream(path);
        Properties properties = new Properties();//创建 properties 变量
        properties.load(inputStream);          //实现属性文件与输入流连接
        //封装数据库表名称作为 map 对象中的 key，字段信息作为 map 对象中的 value
        Object[] keys = properties.keySet().toArray();
        Map map = new HashMap();
        for (int i = 0; i < keys.length; i++) {
            String key = (String) keys[i];
            String value = properties.getProperty(key);
            ParameterObj parameterObj = new ParameterObj();
            parameterObj.setTablName(key);
            parameterObj.setSqlstr(value);
            map.put(key, parameterObj);
        }
        return map;                             //返回 map 对象
    }
}
```

注意：在上述代码中首先获取属性文件流，然后获得属性文件中的数据库表名称与字段，最后返回将表名称作为 key，字段信息作为 value 的 map 对象。

3．sourceTable.xml属性文件

sourceTable.xml 文件用来配置英国和日本数据库相关表格的查询条件，该文件的具体内容如代码 24.9 所示。

代码 24.9　配置查询条件：sourceTable.xml

```
...
<sourceTable>
   <!--对 BSSDATA 进行查询条件 -->
   <BSSDATA>
      select bssid,bssname,createtime,bssdata1,bssdata2,bssdata3 from
      bssdata
      where bssid=#bssid#
   </BSSDATA>
   <!--对 SITEDATA 进行查询条件 -->
   <SITEDATA>
      select siteid,sitename,createtime,sitedata1,sitedata2 from sitedata
     where siteid=#siteid#
   </SITEDATA>
</sourceTable>
```

读取 sourceTable.xml 配置文件的工具类为 ParseSourceXml，该文件的具体内容如代码 24.10 所示。

代码 24.10　读取 sourceTable 配置文件：ParseSourceXml.java

```
...
public class ParseSourceXml {
    //解析 XML 文件，组件 SQL 语句
    public String parserString(String sqlStr, Object obj)
            throws NoSuchMethodException, IllegalAccessException,
            InvocationTargetException, NoSuchFieldException {
```

```
        Class c1 = obj.getClass();
        String[] strs = sqlStr.split("#");
        String sqlstr = "";
        for (int i = 0; i < strs.length; i++) {
            if (i % 2 == 0) {
                sqlstr = sqlstr + strs[i];
            } else {
                String fileTemp = strs[i];
                String para = "get" + fileTemp.substring(0, 1).toUpper-
                Case()+ fileTemp.substring(1, fileTemp.length());
                // 利用 java 反射机制获得相应的方法
                Method method = c1.getDeclaredMethod(para, null);
                Object object = method.invoke(obj, null);
                sqlstr = sqlstr + "'" + object.toString() + "'";
            }
        }
        return sqlstr;
    }
    //解析资源表文件获得查询语句
    public Vector parseXmlContent(String filePath, Object paraObj)
            throws DocumentException, IllegalAccessException,
            NoSuchMethodException, NoSuchFieldException,
            InvocationTargetException {
        // 取配置文件
        ClassLoader classLoader = Thread.currentThread()
                .getContextClassLoader();
        InputStream inputStream = classLoader.getResourceAsStream(file-
        Path);
        SAXReader saxReader = new SAXReader();
        // 取回根接点
        Document document = saxReader.read(inputStream);
        Element element = document.getRootElement();
        List tableElement = element.elements();
        Vector vector = new Vector();
        for (int i = 0; i < tableElement.size(); i++) {
            Element childelement = (Element) tableElement.get(i);
            String sqlstr = parserString(childelement.getTextTrim(),
            paraObj);
            ParameterObj persistenceObj = new ParameterObj();
            // 根据根接点取相应的正文内容
            persistenceObj.setTablName(childelement.getName().trim());
            persistenceObj.setSqlstr(sqlstr);
            vector.add(persistenceObj);
        }
        return vector;
    }
}
```

【代码解析】

在上述代码中首先通过读取配置文件 sourceTable.xml，获取数据库表名称与查询语句。然后将获得的查询语句进行解析分解，利用 Java 反射获得 get()方法，从而取得对象属性的值。最后将获得值重新组成完整的条件查询语句，并且将它们封装到向量中，其中数据库名称是 key，查询语句是值。

类 ParameterObj 用来实现数据库表与该表对应查询语句工具类，即将配置文件中的数据表与该表对应的查询语句赋值给该类的对象，从而用来进行查询数据。该类的具体内容如代码 24.11 所示。

代码 24.11　读取 sourceTable 配置文件：ParameterObj.java

```
...
public class ParameterObj {
    private String tablName;                     //创建表名称属性
    private String sqlstr;                       //创建 SQL 语句属性
    //省略上述属性的 set()和 get()方法
...
}
```

至此，就完成了数据汇聚系统所有相关数据库和表的设计。

24.4　数据汇聚系统具体实现

当系统具体运行时，首先请求会发送给 Struts 2.x 框架中的具体处理器 Action，然后该 Action 类调用 Spring 框架中的 Ioc 容器访问 iBATIS 框架中的 DAO 层，而 iBATIS 框架中的 DAO 层主要负责数据库交互，最后把 DAO 层返回的数据传送给 JSP 页面而响应请求。

24.4.1　DAO 层设计——资源数据库（Englishdata 和 Japandata）

在该系统中 DAO 层分成两部分来设计，资源数据库（Englishdata 和 Japandata）的 DAO 层设计和本地数据库（Americaldata）的 DAO 层设计。而对于资源数据库 DAO 层设计分成两个步骤来实现，它们分别为获取 iBATIS 框架的客户端和实现数据采集器。

1. 获取iBATIS框架客户端

在获取 iBATIS 框架客户端的类 SqlMapClientManager 中，不仅可以读取相应的数据库配置信息，而且还可以实例化 SQLMap 从而获取 sqlMapClient 对象。该文件的具体内容如代码 24.12 所示。

代码 24.12　关于 iBATIS 框架客户端工具类：SqlMapClientManager.java

```
...
public class SqlMapClientManager {
     //通过传入 path 参数得到相应的 Ibatis 核心控制器
    public SqlMapClient getSqlMapClientTemplate(String path)throws IOExce
    ption {
        Reader reader= Resources.getResourceAsReader(path);
        SqlMapConfigParser sqlMapConfigParser=new SqlMapConfigParser();
        SqlMapClient sqlMapClient= sqlMapConfigParser.parse(reader);
        return sqlMapClient;
    }
}
```

2. 实现数据采集器

在具体实现数据采集器时，分成两个步骤来实现：查找资源数据库数据和把数据存储到本地数据库。即在查找资源数据库数据类 RemoteDataRowHandler 中实现多条件查找，

该文件的具体内容如代码 24.13 所示。

代码 24.13　查找工具类：RemoteDataRowHandler.java

```
...
public class RemoteDataRowHandler implements RowHandler {
    //定义临时使用变量
    private String tablName;                    //创建表名称属性
    private String columnMap;                   //创建字段集属性
    private String proid;                       //创建省份标识属性
    private String proname;                     //创建省份名称属性
    private BufferedWriter bufferedWriter;      //创建缓存流属性
    public List list = new ArrayList();         //创建实例化列表类对象属性
    //省略上述属性的 get()和 set()方法
    ...
    //重写 handleRow()方法
    public void handleRow(Object object) {
        try {
            String str = (String) object;       //定义缓存字符串
            InputStream inputStream = null;     //定义输入/输出流
            //利用输入流进行过滤字符
            inputStream = new ByteArrayInputStream(str.getBytes("gb2312"));
            SAXReader saxReader = new SAXReader();   //定义 XML 文件解析器
            Document document = null;                //定义文档解析类
            //将输入文件流读入的 XML 文件解析成树形结构
            document = saxReader.read(inputStream);
            Element element = document.getRootElement();//获得 XML 文件的节点
            List contentElement = element.elements();
                                                //将这些节点封装成 list 对象
            //组建插入数据的语句
            String sqlstr = "insert into " + tablName + columnMap + " values
            " + "(";
            int j = 1;
            ParameterObj pobj = new ParameterObj(); //实例化数据与表映射工具类
            //遍历这些节点后组成完整的插入语句
            for (int i = 0; i < contentElement.size(); i++) {
                Element elementinfo = (Element) contentElement.get(i);
                if (j == contentElement.size()) {
                    sqlstr = sqlstr + "'" + elementinfo.getTextTrim().trim()
                    .replace("'","|") + "'";
                } else {
                    sqlstr = sqlstr + "'" + elementinfo.getTextTrim().trim()
                    .replace("'","|") + "',";
                }
               j++;
            }
            sqlstr = sqlstr +","+"'"+proid+"'"+","+"'"+proname+"'"+")";
            bufferedWriter.write(sqlstr+"\r\n");
            pobj.setSqlstr(sqlstr);
            list.add(pobj);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
}
```

【代码解析】

在上述代码中重写了 iBATIS 框架中的 RowHandler()方法，在该方法中首先通过 iBATIS 的客户端使用组装好的查询条件 SQL 语句获取数据，然后将得到的数据重新封装成多条完整的载值的 SQL 语句，最后将这些 SQL 语句存储到 List 集合中。

RemoteDataTriggerImpl 类实现把利用 RemoteDataRowHandler 类中 SQL 语句查询到的数据存储到本地数据库中，该类的具体内容如代码 24.14 所示。

代码 24.14　存储到本地数据库：RemoteDataTriggerImpl.java

```
...
public class RemoteDataTriggerImpl implements RemoteDataTrigger {
    private Filepath filepath;     //引用在 spring 容器中注入的文件地址属性
    private LocalDataDao dataDao;  //引用在 spring 容器中注入的本地持久层 Dao
    //自动触发获取目标数据库信息并存储到本地数据库
    public void execute(JobExecutionContext jobCtx)throws JobExecutionEx-
    ception{
        //引用 XML 解析资源文件类并实例化
        ParseSourceXml parseSourceXml = new ParseSourceXml();
        //引用我们的公用属性类，并实例化
        ParameterObject pobj = new ParameterObject();
        //设定查询条件
        pobj.setBssid(1);
        pobj.setSiteid(2);
        try {
            //引用公用表列类并实例化
            ParameterColumn parameterColumn = new ParameterColumn();
            //读取数据库表列配置文件，获得 map 类型的数据表列信息
            Map columnmap = parameterColumn
                    .getColumnContent("conf/column.properties");
            //实例化 Ibatis 工具类
            SqlMapClientManager sqlMapClientManager = new SqlMapClientMan-
            ager();
            //通过读取 spring 注入的配置文件信息得到 Ibatis 核心控制工具
            SqlMapClient sqlMapClient = sqlMapClientManager
                    .getSqlMapClientTemplate(filepath.getSqlMappath());
            //通过解析以数据库表名称为节点，包含具体查询语句的配置文件得到包含这些信
            息的向量对象
            Vector vector = parseSourceXml.parseXmlContent("conf/source-
            Table.xml", pobj);
            //实例化多条查找工具类
            RemoteDataRowHandler rhandler = new RemoteDataRowHandler();
            //定义文件输出流
            FileWriter fileWriter = new FileWriter(new File("c:\\ibatis.
            txt"));
            //输入/输出缓冲流
            BufferedWriter bufferedWriter = new BufferedWriter(fileWri-
            ter);
            //将缓冲流中的数据赋值给多条查找工具对象
            rhandler.setBufferedWriter(bufferedWriter);
            //遍历向量中并转型成公用属性类
            for (int i = 0; i < vector.size(); i++) {
                ParameterObj parameterObj = (ParameterObj) vector.get(i);
                //得到相应表的列对应的列信息
                String column = ((ParameterObj) (columnmap.get(parameter-
                Obj
```

```
                    .getTablName()))).getSqlstr();
                //对多条操作对象赋值
                rhandler.setColumnMap(column);
                rhandler.setTablName(parameterObj.getTablName());
                rhandler.setProid(filepath.getProid());
                rhandler.setProname(filepath.getProname());
                //将向量中的数据多条查找
                sqlMapClient.queryWithRowHandler("getRemoteData", parame-
                terObj,
                        rhandler);
            }
            bufferedWriter.close();
            fileWriter.close();
            List list = rhandler.list;
            for (int i = 0; i < list.size(); i++) {
                ParameterObj obj = (ParameterObj) list.get(i);
                //将多条查找封装后的数据插入到本地数据库
                dataDao.insertData(obj);
            }
        } catch (DocumentException e) {
            e.printStackTrace();
        }
    }
}
```

【代码解析】

- 首先设定了 4 个准备条件：通过 bssid=1 与 siteid=2 设定查询条件；读取字段配置文件 column.properties；通过 spring 框架容器加载的 filepath 获得资源数据库 iBATIS 客户端；设定临时文件为 c:\\ibatis.txt。
- 然后通过调用 iBATIS 客户端提供的多条查找方法 queryWithRowHandler()进行资源采集，同时把采集到的资源存储到临时文件里。接着解析临时文件，将数据转存到 ArrayList 数组。
- 最后，通过调用 LocalDataDao 实例，将数据存储到本地数据库里。
- 在编写数据采集器时涉及的两个类 RemoteDataRowHandler 和 RemoteDataTrigger Impl 中，都需要调用 LocalDataDao 类，关于该类的具体内容请看下面内容。

24.4.2　DAO 层设计——本地数据库

在本系统关于本地数据库的 DAO 层中，主要涉及 LocalDataDao 接口和实现类 LocalDataDaoImpl。接口 LocalDataDao 主要用来定义处理本地数据库的各种方法，具体内容如代码 24.15 所示。而实现类 LocalDataDaoImpl 的具体内容如代码 24.16 所示。

代码 24.15　查找资源：LocalDataDao.java

```
...
public interface LocalDataDao {
    public void insertData(ParameterObj parameterObj);
                                                        //向本地数据库插入信息
    public Object loginUser(Object obj)throws IOException;
                                                         //管理员登录
    public List selectBssData() throws IOException;     //遍历 bssdata
    public List selectSiteData()throws IOException;     //遍历 sitedata
}
```

【代码解析】

- ❑ insertData()方法用来实现保存数据功能。
- ❑ loginUser()方法用来实现登录功能。
- ❑ selectBssData()方法用来实现遍历数据库表格 bssdata。
- ❑ selectSiteData()方法用来实现遍历数据库表格 sitedata。

代码 24.16　查找资源：LocalDataDaoImpl.java

```
...
//继承 SqlMapClientDaoSupport，使用 Ibatis 进行持久化操作
public class LocalDataDaoImpl extends SqlMapClientDaoSupport implements
LocalDataDao{
    public void insertData(ParameterObj parameterObj) {
                                            //实现向本地数据库中插入信息
        getSqlMapClientTemplate().insert("insertData", parameterObj);
    }
    public List selectBssData() throws IOException {
                                            //实现遍历 bssdata()方法
        ParameterObject p = new ParameterObject();
         List list=getSqlMapClientTemplate().queryForList("selectBss-
        Data", p);
         if(list==null){
            list=new ArrayList();
         }
        return list;
    }
    public List selectSiteData() {          //实现遍历 sitedata()方法
        ParameterObject p = new ParameterObject();
        List list= getSqlMapClientTemplate().queryForList("selectSite-
        Data", p);
        return list;
    }
    public Object loginUser(Object obj) throws IOException {
                                            //实现管理员登录方法
        UserInfoVo user=(UserInfoVo)obj;
        System.out.println("userno"+user.getUsername());
        return getSqlMapClientTemplate().queryForObject("loginUser",user);
    }
}
```

【代码解析】

- ❑ loginUser()方法主要用来处理管理员登录信息，即以管理员身份登录时提供的用户名作为查询条件，查询数据表格 userinfo 中的记录，如果存在该记录，则返回数据，否则返回空。
- ❑ insertData()方法主要用来处理将采集回来的数据，存储到美国本地数据库相应的表中。selectSiteData()方法主要用来处理本地数据库表格 sitedata，即遍历美国数据库 sitedata 中的数据。
- ❑ selectBssData()方法主要用来处理本地数据库表格 bssdata，即遍历美国数据库中 bssdata 表的数据。
- ❑ 该类之所以要继承 SqlMapClientDaoSupport 类，因为需要 Spring 框架封装 iBATIS

框架的客户端，同时 SqlMapClientDaoSupport 类也是 Spring 框架整合 iBATIS 框架以后提供给外界使用的 API，最后还可以通过 SqlMapClientDaoSupport 类的 SqlMapClientTemplate()方法实现对数据进行持久化操作。

24.4.3　Spring 框架配置信息

在本系统中有两个 Spring 框架的配置文件，Spring 应用上下文配置文件 applicationContext.xml 和 Spring 数据库上下文 datalocalContext.xml。

Spring 应用上下文配置文件 applicationContext.xml 的具体内容，如代码 24.17 所示。

代码 24.17　应用上下文配置文件：applicationContext.xml

```
...
   <!--配置 spring 框架管理事务 -->
   <bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransaction-
      Manager">
      <property name="dataSource" ref="dataSource"/>
   </bean>
   <bean id="OrderService"
      class="org.springframework.transaction.interceptor.TransactionPr-
      oxyFactoryBean">
      <property name="transactionManager">
         <ref local="transactionManager"></ref>
      </property>
      <property name="target">
        <ref bean="remoteDataCollection"/>
      </property>
      <property name="transactionAttributes">
         <props>
            <prop key="insert*">PROPAGATION_REQUIRED</prop>
         </props>
      </property>
   </bean>
    <!--注入自动采集数据类的 Dao 与 filepath 属性-->
    <bean id="remoteTrigger"
        class="com.cjg.remote.persistence.RemoteDataTriggerImpl">
        <property name="dataDao">
            <ref bean="dataDao" />
        </property>
        <property name="filepath">
            <ref bean="filepath1" />
        </property>
    </bean>
    <!--为 filepath1 配置相应信息，包括资源文件地址与属性值-->
    <bean id="filepath1" class="com.cjg.utils.Filepath">
        <property name="sqlMappath" value="conf/sourceData1.xml" />
        <property name="proid" value="01" />
        <property name="proname" value="English" />
    </bean>
    <!--注入自动采集数据类的 Dao 与 filepath 属性-->
    <bean id="remoteTrigger2"
        class="com.cjg.remote.persistence.RemoteDataTriggerImpl">
        <property name="dataDao">
            <ref bean="dataDao" />
```

```
        </property>
        <property name="filepath">
            <ref bean="filepath2" />
        </property>
    </bean>
    <!--为 filepath2 配置相应信息，包括资源文件地址与属性值-->
    <bean id="filepath2" class="com.cjg.utils.Filepath">
        <property name="sqlMappath" value="conf/sourceData2.xml" />
        <property name="proid" value="02" />
        <property name="proname" value="Japan" />
    </bean>
    <!--配置 spring 自动触发的类与方法-->
    <bean id="remoteMethodTrigger1"
        class="org.springframework.scheduling.quartz.MethodInvokingJobD-
        etailFactoryBean">
        <property name="targetObject" ref="remoteTrigger" />
        <property name="targetMethod" value="execute" />
        <property name="concurrent" value="false" />
    </bean>
    <bean id="remoteMethodTrigger2"
        class="org.springframework.scheduling.quartz.MethodInvokingJob
        DetailFactoryBean">
        <property name="targetObject" ref="remoteTrigger2" />
        <property name="targetMethod" value="execute" />
        <property name="concurrent" value="false" />
    </bean>
    <!--为 remoteMethodTrigger1 配置触发策略，在每天的凌晨 1 点自动触发-->
    <bean id="remoteMethodCron"
        class="org.springframework.scheduling.quartz.CronTriggerBean">
        <property name="jobDetail" ref="remoteMethodTrigger2" />
        <property name="cronExpression" value="0 0 1 * * ?" />
    </bean>
    <bean id="remoteMethodCron1"
        class="org.springframework.scheduling.quartz.CronTriggerBean">
        <property name="jobDetail" ref="remoteMethodTrigger1" />
        <property name="cronExpression" value="0 0 1 * * ?" />
    </bean>
    <!--为 spring 配置自动触发工厂-->
    <bean id="remoteDataCollection" autowire="no"

    class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <property name="triggers">
            <list>
                <ref bean="remoteMethodCron" />
                <ref bean="remoteMethodCron1" />
            </list>
        </property>
    </bean>
</beans>
```

【代码解析】

- 首先通过 id 为 transactionManager 的<bean>配置关于 Spring 框架事务管理。
- 接着通过 id 为 remoteDataCollection 的<bean>为该系统配置两个自动触发工厂：remoteMethodCron 和 remoteMethodCron1。然后配置这两个触发工厂的触发策略：remoteMethodCron 触发工厂的 DAO 层数据采集方法为 remoteMethodTrigger2，触发时间为凌晨 1 点；remoteMethodCron1 触发工厂的 DAO 层数据采集方法为 remoteMethodTrigger1，触发时间为凌晨 1 点。

- 最后为数据采集方法 remoteMethodTrigger1 配置数据区别标识：remoteTrigger；数据采集方法 remoteMethodTrigger2 配置数据区别标识：remoteTrigger2。最后通过 id 为 remoteTrigger 和 filepath1 的<bean>为采集到的数据添加区别标识，对数据源 sourcedata 的属性 proid 值为 1 而属性 proname 值为 English；通过 id 为 remoteTrigger2 和 filepath2 的<bean>为采集到的数据添加区别标识，对数据源 sourcedata 的属性 proid 值为 2 而属性 proname 值为 Japan。

Spring 数据库上下文 datalocalContext.xml 的具体内容如代码 24.18 所示。

代码 24.18　数据库上下文配置文件：datalocalContext.xml

```
...
<beans>
   <!--配置数据库配置属性文件-->
   <bean id="propertyConfigurer" class="org.springframework.beans.fact-
   ory.config.PropertyPlaceholderConfigurer">
      <property name="location">
         <value>/WEB-INF/localconf/jdbc.properties</value>
      </property>
   </bean>
   <!--配置数据源-->
   <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
   destroy-method="close">
      <property name="driverClassName">
         <value>${jdbc.driverClassName}</value>
      </property>
      <property name="url">
         <value>${jdbc.url}</value>
      </property>
      <property name="username">
         <value>${jdbc.username}</value>
      </property>
      <property name="password">
         <value>${jdbc.password}</value>
      </property>
      <property name="maxActive" value="100"/>
      <property name="maxIdle" value="30"/>
      <property name="maxWait" value="1000"/>
      <property name="defaultAutoCommit" value="true"/>
      <property name="removeAbandoned" value="true"/>
      <property name="removeAbandonedTimeout" value="60"/>
   </bean>
   <!--配置映射文件-->
   <bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMap-
   ClientFactoryBean">
      <property name="dataSource">
         <ref bean="dataSource"/>
      </property>
      <property name="configLocation">
         <value>/WEB-INF/localconf/sql-map-config.xml</value>
      </property>
   </bean>
   <!--注入本地 Dao-->
   <bean id="dataDao" class="com.cjg.local.persistence.LocalDataDaoImpl">
     <property name="sqlMapClient">
        <ref bean="sqlMapClient"/>
     </property>
```

```
    </bean>
</beans>
```

【代码解析】

- 通过 id 为 propertyConfigurer 的<bean>配置数据库的属性文件。
- 为本地数据库配置 iBATIS 框架客户端各种参数：包含 Spring 框架整合 iBATIS 的对象工厂、数据源、映射文件。

在上述代码中实现引用 jdbc.properties 属性文件配置，在该属性文件中实现了对本地数据库的驱动、连接地址、本地数据库用户名和密码配置。该文件的具体内容如代码 24.19 所示。

代码 24.19　本地数据库配置文件：jdbc.properties

```
#数据库驱动
jdbc.driverClassName=com.microsoft.jdbc.sqlserver.SQLServerDriver
#数据库 URL 地址
jdbc.url=jdbc:microsoft:sqlserver://127.0.0.1:1433;DatabaseName=America
ldata
#用户名
jdbc.username=sa
#密码
jdbc.password=root
```

24.4.4　业务层逻辑设计

当系统具体运行时，首先请求会发送给 Struts 2.x 框架中的具体处理器 Action，然后该 Action 类调用 Spring 框架中的 Ioc 容器访问 iBATIS 框架中的 DAO 层，而 iBATIS 框架中的 DAO 层主要负责数据库交互。本章将详细讲解 iBATIS 框架中的 DAO 层。

接口 LocalDataFacade 用来定义各种与数据库进行交互的方法，该接口的具体内容如代码 24.20 所示。

代码 24.20　操作数据库：LocalDataFacade.java

```
public interface LocalDataFacade {
    public void insertData(ParameterObj parameterObj);
                                                  //插入数据方法
    public Object loginUser(Object obj)throws IOException;
                                                  //管理用户方法
    public List selectBssData() throws IOException; //遍历 bss 数据库方法
    public List selectSiteData()throws IOException; //遍历 site 数据库方法
}
```

【代码解析】

- insertData()方法用来实现保存数据功能。
- loginUser()方法用来实现登录功能。
- selectBssData()方法用来实现遍历数据库表格 bssdata。
- selectSiteData()方法用来实现遍历数据库表格 sitedata。

实现类 LocalDataFacadeImpl 用来实现各种与数据库进行交互的方法，该类的具体内容

如代码 24.21 所示

代码 24.21　操作数据库：LocalDataFacadeImpl.java

```
public class LocalDataFacadeImpl implements LocalDataFacade{
    LocalDataDao localData;                      //创建属性 localData
    public LocalDataDao getLoacalData() {        //配置属性 localData
        return localData;
    }
    public void setLoacalData(LocalDataDao loaclData) {
        this.localData = loaclData;
    }
    public void insertData(ParameterObj parameterObj) {
                                                 //实现 insertData()方法
        localData.insertData(parameterObj);
    }
    public List selectBssData() throws IOException {
                                                 //实现 selectBssData()方法
        return localData.selectBssData();
    }
    public List selectSiteData()throws IOException {
                                                 //实现 selectSiteData()方法
        return localData.selectSiteData();
    }
    public Object loginUser(Object obj) throws IOException {
                                                 //实现 loginUser()方法
        return localData.loginUser(obj);
    }
}
```

接着在 applicationContext.xml 文件中对该程序进行配置。

```
<!--对 LocalDataFacadeImpl 类进行配置 -->
<bean id="localDataFacade" class="com.cjg.local.service.LocalDataFacade-
Impl">
    <property name="loacalData">
        <ref bean="dataDao"/>
    </property>
</bean>
```

注意：在上述代码中，通过调用持久层的方法来实现数据汇聚系统的业务逻辑。

24.5　数据汇聚系统具体实现——表示层

在数据汇聚系统中，会结合 Struts 2.x 框架来实现页面的跳转。该系统涉及的页面有：登录页面 login.jsp、欢迎页面 welcome.jsp、显示 BSS 数据库列表页面 selectBssdata.jsp，以及显示 SITE 数据库列表页面 selectSitedata.jsp。

24.5.1　处理请求 Action 类

在数据汇聚系统中由 Struts 2.0 框架实现页面的跳转，所以在该系统中不仅需要创建

struts.xml 文件，而且还需要创建名为 LocalDataAction 的 Action 类。

在 Action 类 LocalDataAction 中，不仅会验证管理员登录，而且还会处理所有的请求。该文件的具体内容如代码 24.22 所示。

代码 24.22　实现页面跳转：LocalDataAction.java

```
...
public class LocalDataAction extends ActionSupport {
    LocalDataFacade localDataFacade;        //创建 localDataFacade 属性
    List lbssdata=new ArrayList();          //创建 lbssdata 属性
    List lsitedata=new ArrayList();         //创建 lsitedata 属性
    private String username;                //创建 username 属性
    private String password;                //创建 password 属性
    //省略上述属性的 get()和 set()方法
...
    //查询本地数据库中 bssdata 数据库信息
    public String selectBssdata() throws IOException, SQLException {
         lbssdata = localDataFacade.selectBssData();
                                            //遍历名为 bssdata 数据库
        return "bssdata";
    }
    //查询本地数据库中 sitedata 数据库信息
    public String selectSitedata() throws IOException{
        lsitedata = localDataFacade.selectSiteData();
                                            //遍历名为 sitedata 数据库
        return "sitedata";
    }
    //验证用户验证
    public String loginUser() throws IOException {
        //判断 username 和 password 两变量的值
        if (username != null && password != null) {
                                    //当 username 和 password 两变量值不为空
            UserInfoVo user1 = new UserInfoVo();
            user1.setUsername(username);
            UserInfoVo user = (UserInfoVo) localDataFacade.loginUser-
            (user1);
            if (!(user.getUsername().equals(username))) {
                                    //当 username 变量值不相同时
                addFieldError("userno","用户编号不存在！");
            } else if(!(password.equals(user.getPassword()))){
                                    //当 password 变量值不相同时
                addFieldError("password","密码不正确");
            }
        }else {
            addFieldError("userno","请填写用户编号！");
            addFieldError("password","请填写密码！");
        }
        if(hasErrors()){                    //当存在错误时
            return "input";
        }else{                              //当成功时
            return "loginSuccess";
        }
    }
}
```

【代码解析】

- 在查询本地数据库中 bssdata 数据库的方法 selectBssdata()中，通过调用 DAO 层的查询 BSS 数据库信息的方法，得到该数据库列表对象。
- 在查询本地数据库中 sitedata 数据库的方法 selectSitedata()中，通过调用 DAO 层的查询 SITE 数据库信息的方法，得到该数据库列表对象。
- 在实现验证管理员登录 loginUser()中，首先会把关于 username 和 password 的请求实例化成 UserInfoVo 类对象，然后调用 DAO 层验证管理员登录方法，当查询到的密码与 password 相同时，则返回字符串 loginSuccess；否则返回字符串 input。

接着在 applicationContext.xml 文件中对该程序进行配置。

```
<!--对 LocalDataAction 类进行配置 -->
<bean id="dao" class="com.cjg.local.action.LocalDataAction">
    <property name="localDataFacade">
        <ref bean="localDataFacade" />
    </property>
</bean>
```

接着在 struts.xml 文件中配置请求。

```
...
<struts>
    <constant name="struts.i18n.encoding" value="GBK" />
    <constant name="struts.custom.i18n.resources"
        value="globalMessages" />
    <package name="default" extends="struts-default">
        <!--处理用户登录-->
        <action name="loginUser" class="dao"
            method="loginUser">
            <result name="loginSuccess">/jsp/welcome.jsp</result>
            <result name="input">jsp/login.jsp</result>
        </action>
        <!--处理查询 bss 数据请求-->
        <action name="selectBssdata" class="dao"
            method="selectBssdata">
            <result name="bssdata">/jsp/selectBssdata.jsp</result>
        </action>
        <!--处理查询 site 数据请求-->
        <action name="selectSitedata" class="dao"
            method="selectSitedata">
            <result name="sitedata">/jsp/selectSitedata.jsp</result>
        </action>
    </package>
</struts>
```

【代码解析】

在上述 struts.xml 配置文件中不仅会处理 3 种请求：loginUser.action、selectBssdata.action 和 selectSitedata.action，而且还配置了两种 result 属性：input 和 loginSuccess。

24.5.2　各种功能页面

在数据汇聚系统中主要存在 4 个功能页面，login.jsp 页面主要提供管理员登录系统功能，welcome.jsp 页面为该系统的主界面，selectBssdata.jsp 页面实现查看表格 bss 的所有记

录，以及 selectSitedata.jsp 页面实现查看表格 site 的所有记录。

1．设计登录页面login.jsp

页面 login.jsp 用来实现用户登录功能，该页面的具体内容如代码 24.23 所示。

代码 24.23　用户登录页面：login.jsp

```
...
<body>
    <table>
        <td>数据采集管理系统></td>
            <form  id="form" method="post" name="form" action="loginUser.
            action">
                <!--用户名输入框-->
                <td> <s:textfield name="username" label="用户名"  size="25"
                    maxlength="40" /></td>
                <!--密码输入框-->
                <td><s:password  name="password" label="密  码" size="25"
                     maxlength="40" /></td>
                <!--登录按钮-->
                <td><input src="img/login/land.gif" name="submit" alt="登
                录"  type="image"
                    d="submit"/></td>
            </form>
    </table>
</body>
...
```

【代码解析】

当用户进行登录时，就会触发 loginUser.action 请求进行登录验证。如果登录成功，则会跳转到欢迎页面；否则就会返回登录页面。

2．设计欢迎页面welcome.jsp

页面 welcome.jsp 为数据汇聚系统的主界面，该页面的具体内容如代码 24.24 所示。

代码 24.24　系统主界面：welcome.jsp

```
...
<body topmargin="5">
<table >
    <td>欢迎使用数据采集系统</td>
    <form action="newsController.do?flag=3" method="post" name="frm1" >
          <!--查看 BSS 数据链接-->
          <TD><s:a href="selectBssdata.action"><h3>查看 BSS 数据</h3></s:a>
          </a>
          </div></TD>
          <!--查看 SITE 数据链接-->
          <TD><s:a href="selectSitedata.action"><h3>查看 SITE 数据</h3>
          </s:a></a>
          </div></TD>
    </form>
</table>
</body>
...
```

【代码解析】

当用户成功登录后，在欢迎页面中单击数据信息链接时，就会触发 selectBssdata.action 或 selectSitedata.action 这两种请求。

3. 设计查看页面selectBssdata.jsp

页面 selectBssdata.jsp 用来实现查看表格 bss 的全部记录，该页面的具体内容如代码 24.25 所示。

代码 24.25　查看 bss 表格数据页面：selectBssdata.jsp

```
...
<body>
<table>
        <!--标题 -->
        <TD>bbs 数据详情</h1></TD>
        <!--表格头标题 -->
        <TD >bssid</TD>
        <TD > bssname </TD>
        <TD > createtime </TD>
        <TD > bssdata1 </TD>
        <TD > bssdata2 </TD>
        <TD >国家编号</TD>
        <TD >名称</TD>
        <!--遍历 site 数据-->
        <s:iterator value="lbssdata">
            <!--输出 ID-->
            <TD ><s:property value="bssid" /></TD>
            <!--输出名字-->
            <TD ><s:property value="bssname" /></TD>
            <!--输出创建时间-->
            <TD ><s:property value="createtime" /></TD>
            <!--输出 sitedata1 数据-->
            <TD ><s:property value="bssdata1" /></TD>
            <!--输出 sitedata2 数据-->
            <TD ><s:property value="bssdata2" /></TD>
            <!--输出国家编号-->
            <TD ><s:property value="proid"/></TD>
            <!--输出国家名称-->
            <TD ><<s:property value="proname" /></TD>
        </s:iterator>
</table>
</body>
...
```

【代码解析】

当用户成功登录后，在欢迎页面中单击查看 bss 数据信息链接时，就会调用 DAO 层的查询 bss 数据信息的方法，将得到的列表对象传递到显示所有信息列表页面 selectBssdata.jsp。

4. 设计查看页面selectSitedata.jsp

页面 selectSitedata.jsp 用来实现查看表格 site 的全部记录，该页面的具体内容如代码

24.26 所示。

代码 24.26　查看 site 表格数据页面：selectSitedata.jsp

```
...
<body>
<table>
        <!--标题 -->
        <TD>site 数据详情</h1></TD>
        <!--表格头标题 -->
        <TD >siteid</TD>
        <TD > sitename </TD>
        <TD > createtime </TD>
        <TD > sitedata1 </TD>
        <TD > sitedata2 </TD>
        <TD >国家编号</TD>
        <TD >名称</TD>
        <!--遍历 site 数据-->
        <s:iterator value="lsitedata">
            <!--输出 ID-->
            <TD ><s:property value="siteid" /></TD>
            <!--输出名字-->
            <TD ><s:property value="sitename" /></TD>
            <!--输出创建时间-->
            <TD ><s:property value="createtime" /></TD>
            <!--输出 sitedata1 数据-->
            <TD ><s:property value="sitedata1" /></TD>
            <!--输出 sitedata2 数据-->
            <TD ><s:property value="sitedata2" /></TD>
            <!--输出国家编号-->
            <TD ><s:property value="proid"/></TD>
            <!--输出国家名称-->
            <TD ><<s:property value="proname" /></TD>
        </s:iterator>
</table>
</body>
...
```

【代码解析】

当用户成功登录后，在欢迎页面中单击查看 site 数据信息链接时，就会调用 DAO 层的查询 site 数据信息的方法，将得到的列表对象传递到显示所有信息列表页面 selectSitedata.jsp。

24.6 小　　结

本章主要介绍一个完整的数据汇聚系统，该系统实现了经典 MVC 模式，基于 Struts 2.x+Spring+iBATIS 框架构建而成。在具体实现数据汇聚系统时，通过 Spring 框架提供的依赖注入，结合灵巧的 iBATIS 持久层和动态的 XML 解析，达到数据汇聚系统的良好性能。

第 25 章　投票管理系统（Struts 2.x+Spring+Hibernate）

在第 16 章虽然实现了网上投票功能，但是在具体管理时却十分不方便。例如当增加新投票项目、修改原有投票信息等。为了使投票管理系统更具有维护性、灵活性和完整性，还需要投票信息管理功能。

本章将通过 Struts 2.x+Spring+Hibernate 框架技术来实现一个完整的投票管理系统，该系统完全符合 Java EE 开发标准的 4 层结构体系，数据库管理系统则为 MySQL AB 公司的 MySQL 5.0。

25.1　投票管理系统简述

本节的投票管理系统由前台和后台两部分组成：前台用来让用户执行投票操作，显示投票结果，后台则是对投票和管理员信息进行管理。该系统在结构上主要分成领域模型层、业务层、持久层和表示层，功能模块如图 25.1 所示。该系统的目录机构如图 25.2 所示。

25.1.1　投票管理系统功能描述

在本章的投票管理系统中，通过投票管理功能，可以完成创建新投票项、修改原有投票信息、删除投票项、查找特定投票项和验证用户权限功能。通过网上投票功能，可以完成对投票项的投票，显示投票结果。

在验证用户权限的模块中，只有具有管理权限的用户才能登录后台模块。当管理员登录后台成功后，如果想修改自己的密码，则可以登录进入修改密码的页面。

在创建新投票项的模块中，当管理员登录后台成功后，打开修改投票页面单击“新增一个投票选项”链接就可以在出现的“增加新投票项”页面中实现增加新投票项功能。

在修改原有投票信息的模块中，当管理员登录后台成功后，打开修改投票页面单击“更新”链接，就可以在出现的“更新投票项”页面中实现更新投票项功能。

在删除投票信息的模块中，当管理员登录后台成功后，打开修改投票页面单击相应投票项的“删除”链接就可以实现删除投票项功能。

在查找特定投票的模块中，当管理员登录后台成功后，在打开查找投票页面中输入完整或部分的投票主题并提交，就可以实现查找特定投票信息。

在网上投票功能中，只要浏览者（任何）进入投票页面，在系统提供的投票主题及投票内容中进行投票后，系统就会自动显示投票结果。

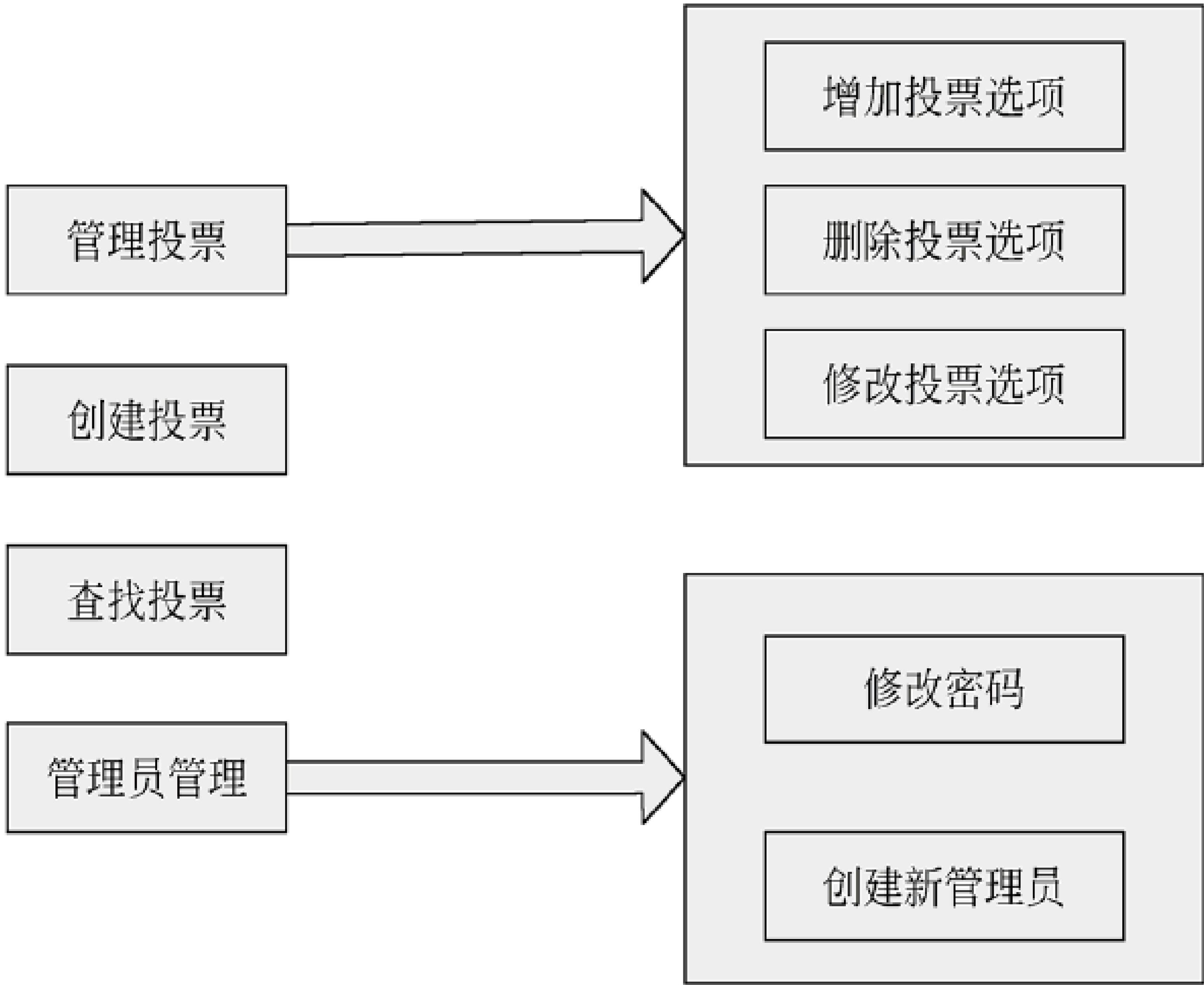


图 25.1　投票管理系统工作原理

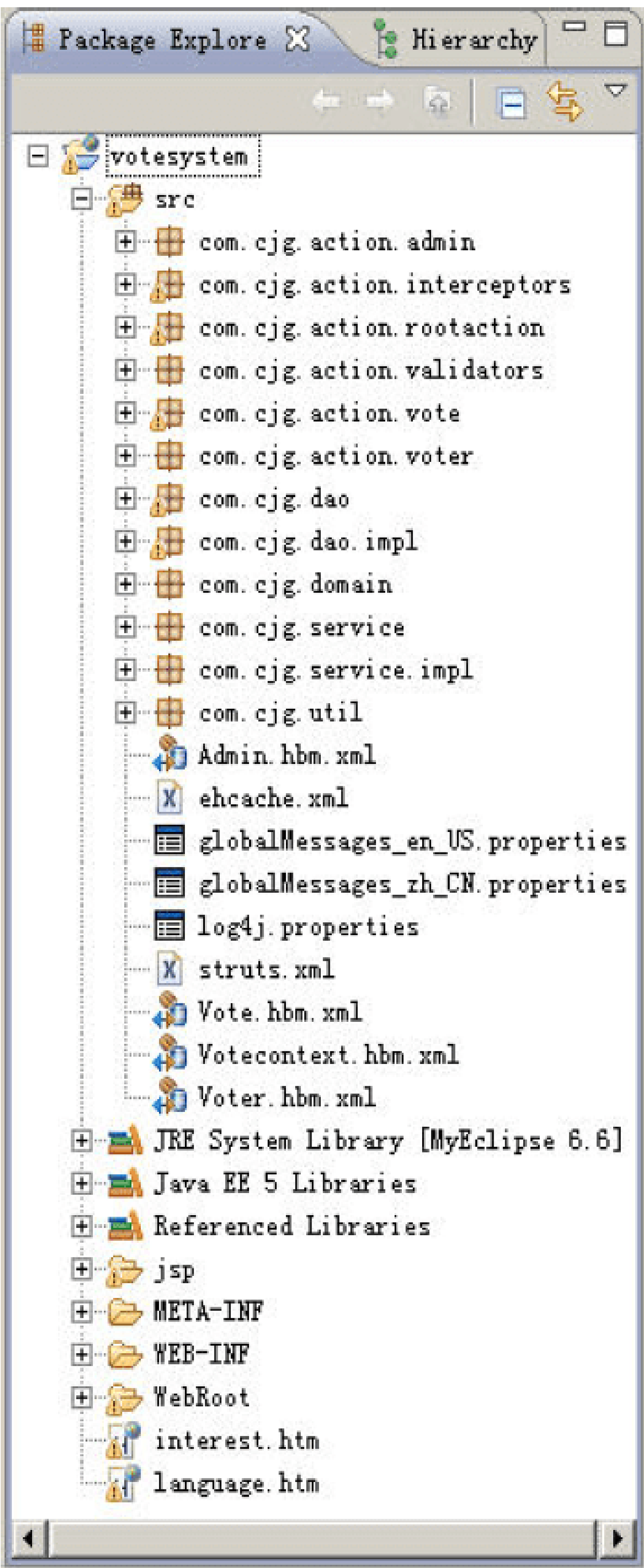


图 25.2　目录结构

25.1.2　投票管理系统操作流程

在本节中，以直观的方式来向读者介绍整个投票管理系统要实现的功能。这些功能包括投票管理功能和网上投票功能两大部分。

1．管理员登录

通过网址 http://localhost:8080/votesystem/login.jsp 打开登录页面后（如图 25.3 所示），在该页面中填写相应的信息后，就会进入系统主界面（如图 25.4 所示）。



图 25.3　登录页面

2．创建投票

在系统主界面中，通过“投票主页”|“创建投票”按钮就可以打开“创建新投票”页面（如图 25.5 所示），在该页面中填写相应的信息后，单击“下一步”按钮就会进入“输入投票选项内容”页面（如图 25.6 所示）。填写完“输入投票选项内容”页面后，单击“下一

步”按钮就会进入“编辑投票信息”页面（如图 25.7 所示），在该页面中可以对已经创建好的投票信息进行修改。通过单击“编辑投票信息”页面中的“更新”按钮，就可以进入显示管理投票信息的页面（如图 25.8 所示）。在该页面中单击“编辑”链接就可以转入“编辑投票信息”页面进行修改。



图 25.4　系统主界面

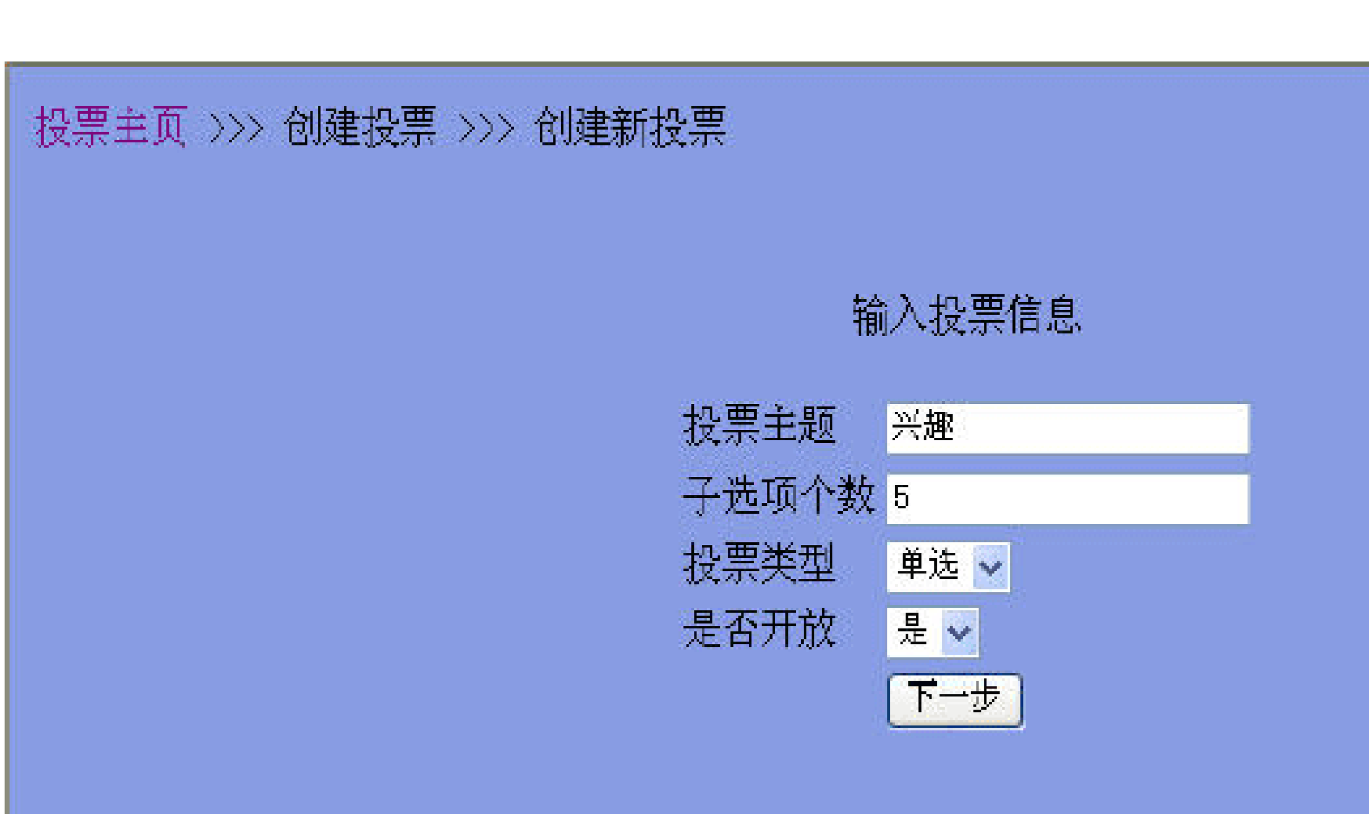


图 25.5　输入投票信息

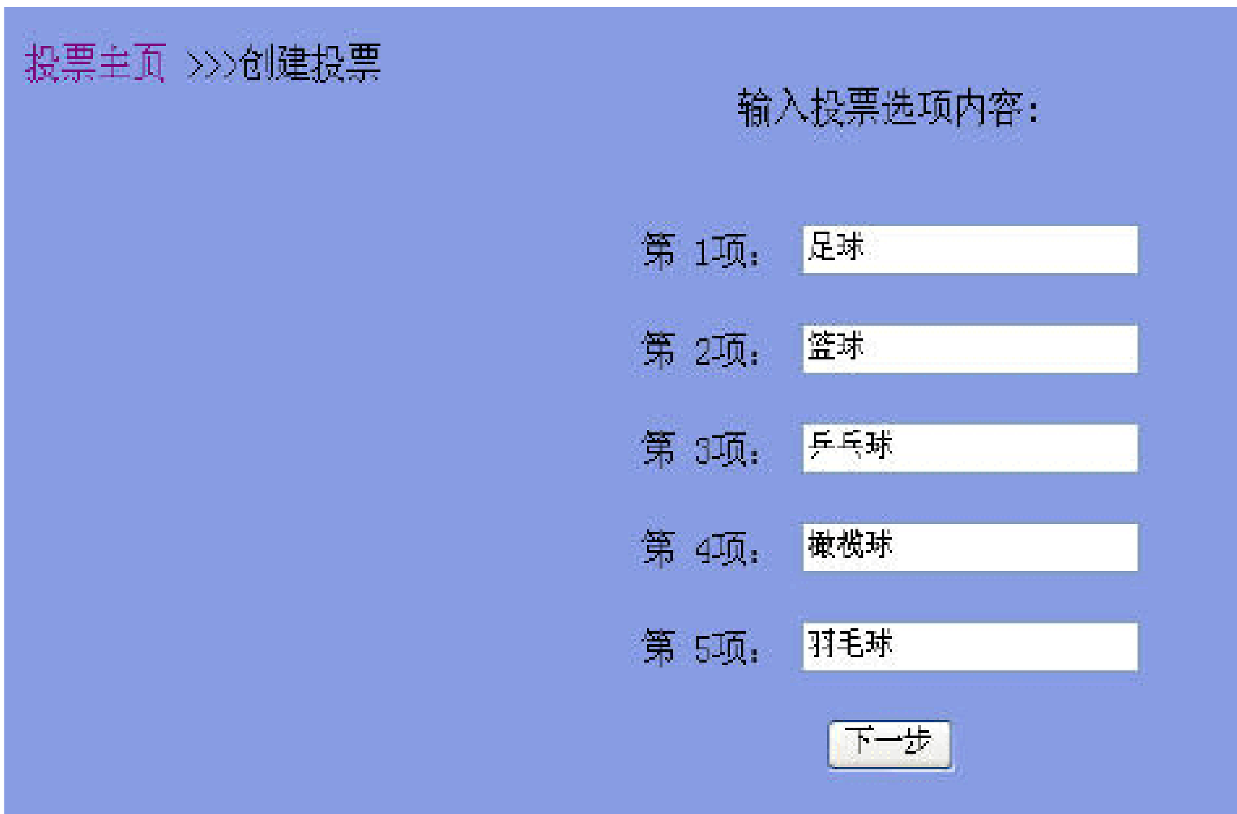


图 25.6　输入投票选项内容

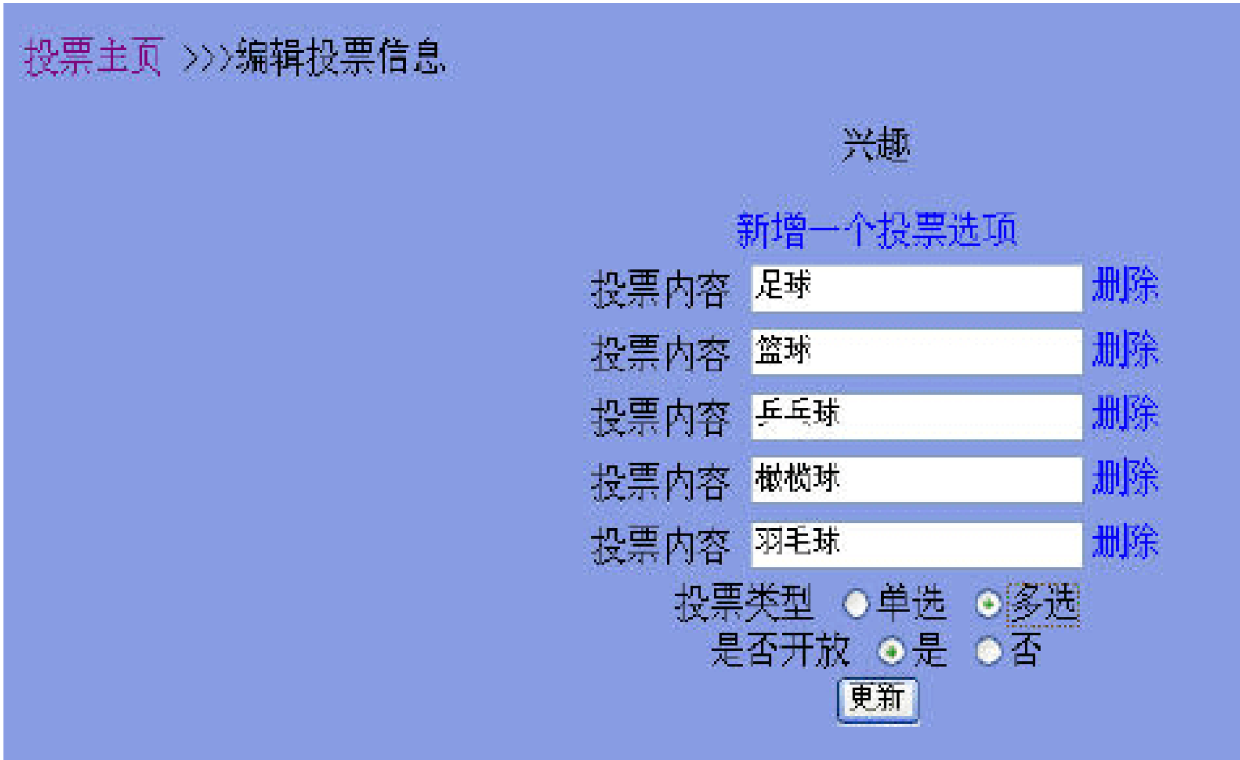


图 25.7　编辑投票信息



图 25.8　管理投票

通过上述步骤再创建一个“语言”投票，最后的管理投票页面如图 25.9 所示。

投票主页 >>>管理投票

投票编号	投票主题	创建时间	创建人	投票状态	投票类型	编辑
1	兴趣	2009-06-21 18:40:34	cjgong	开放	多选	编辑
2	语言	2009-06-21 18:44:34	cjgong	开放	单选	编辑

图 25.9　投票列表

3．查找投票

在系统主界面中，通过单击“投票主页”|“查找投票”按钮就可以打开“查找投票”页面（如图 25.10 所示）。在该页面中首先填写“语言”，然后单击“确定”按钮就可以转入关于该投票选项的管理投票页面（如图 25.11 所示）。

投票主页 >>>查找投票

输入查询内容 语言 确定

图 25.10　查找投票页面

投票主页 >>>管理投票

投票编号	投票主题	创建时间	创建人	投票状态	投票类型	编辑
2	语言	2009-06-21 18:44:34	cjgong	开放	单选	编辑

图 25.11　语言管理投票页面

4．修改密码

在系统主界面中，通过单击“投票主页”|“修改密码”按钮就可以打开“修改密码”页面（如图 25.12 所示）。在该页面中填写相应的内容后就可以转入密码修改成功页面（如图 25.13 所示）。

投票主页 >>> 管理员管理 >>> 修改密码

修改管理员密码

输入旧密码 123456
输入新密码 111111
再次输入新密码 111111
确定 重置

图 25.12　修改密码页面



图 25.13　密码修改成功页面

至此，就完成了投票管理功能的演示。下面将演示如何实现网上投票功能。通过网址 http://localhost:8080/votesystem/language.htm 可以打开关于语言投票页面(如图 25.14 所示)。在该页面中选中“汉语”单选按钮并单击“下一步”按钮后，就会转入显示投票结果的页面，如图 25.15 所示。

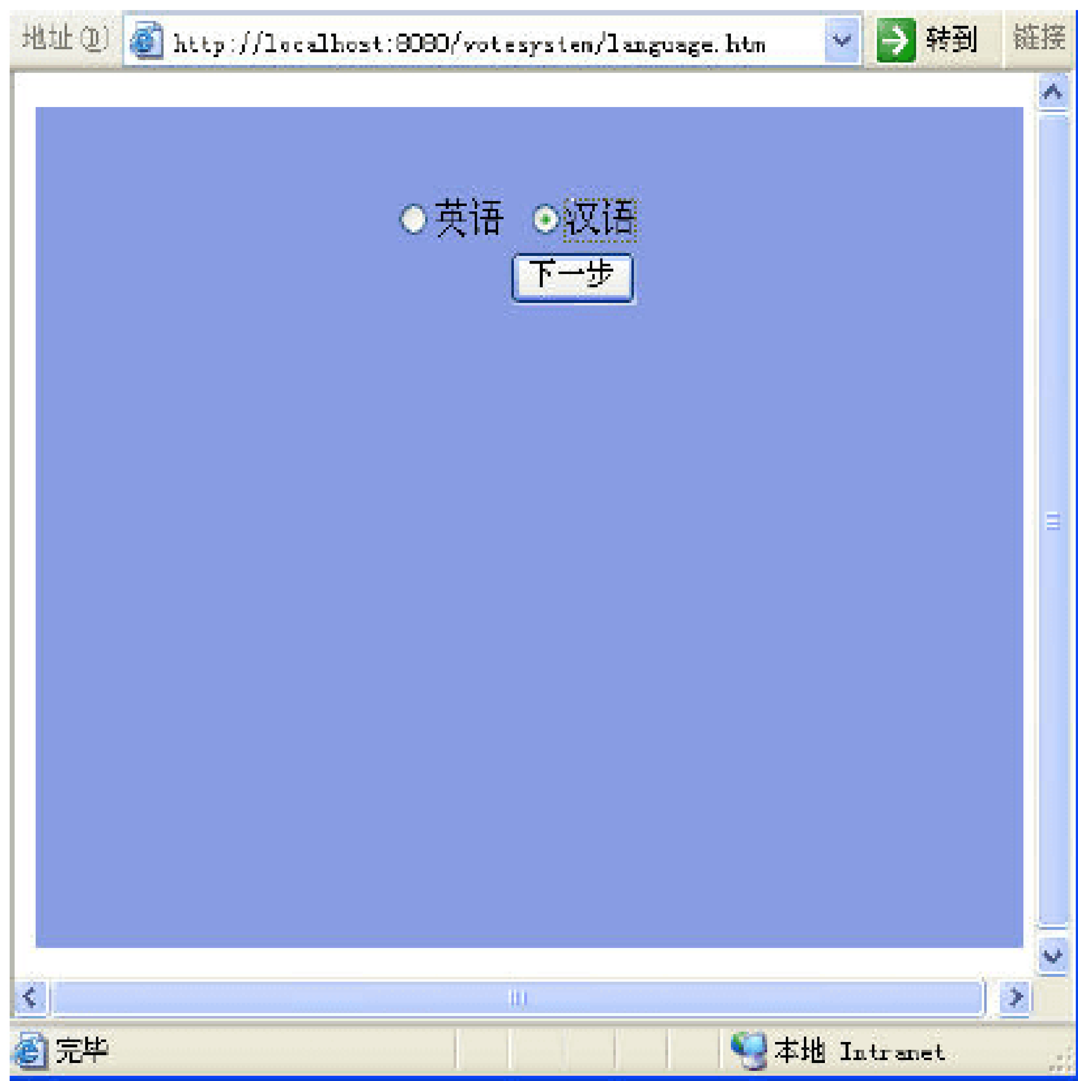


图 25.14　语言投票页面



图 25.15　显示投票结果

如果想查看关于兴趣投票情况，可以通过网址 http://localhost:8080/votesystem/interest.htm 打开“兴趣”投票页面，如图 25.16 所示。



图 25.16　兴趣投票页面

25.2　投票管理系统前期准备

本节除了将详细介绍如何设计关于投票管理系统的数据库和表格外，还将配置实现该

系统将利用的 Struts 2.x+Spring+Hibernate+MySQL 框架的环境。其中 Struts 2.x 框架的版本号为 Struts 2.0、Spring 框架的版本号为 Spring 2.0、Hibernate 框架的版本号为 Hibernate 3.0、数据库 MySQL 为 MySQL 5.0。

25.2.1　数据库设计

投票管理系统需要建立一个数据库并在该数据库中建立 4 张表，存放表格的数据库 votesys、存放管理员信息的表 admin、存放投票主题信息的表 vote、存放投票人信息的表 voter，以及存放投票选项信息的表 votecontext。

1. 创建数据库votesys

如果想创建出数据库 votesys，可以在 MySQL 的命令行窗口中输入如下命令：

```
CREATE DATABASE 'votesys'
```

2. 创建表admin

如果想创建出表 admin，可以在 MySQL 的命令行窗口中输入相关命令。该表的具体信息如表 25.1 所示。

表 25.1　表admin信息

字 段 名 称	数 据 类 型	字 段 说 明
admin_id	int(11)	编号
name	varchar(50)	用户名
password	varchar(50)	密码
logintime	varchar(50)	登录时间

3. 创建表vote

如果想创建出表 vote，可以在 MySQL 的命令行窗口中输入相关命令。该表的具体信息如表 25.2 所示。

表 25.2　表vote信息

字 段 名 称	数 据 类 型	字 段 说 明
vote_id	int(11)	编号
title	varchar(50)	投票主键
createdate	varchar(50)	创建时间
type	int(11)	投票类型
publish	int(11)	是否开发
admin_id	int(11)	编号

4. 创建表voter

如果想创建出表 voter，可以在 MySQL 的命令行窗口中输入相关命令。该表的具体信息如表 25.3 所示。

表 25.3　表voter信息

字 段 名 称	数 据 类 型	字 段 说 明
Id	int(11)	编号
vote_id	int(11)	编号
Ip	varchar(45)	投票人的 ip

5．创建表votecontext

如果想创建出表 votecontext，可以在 MySQL 的命令行窗口中输入相关命令。该表的具体信息如表 25.4 所示。

表 25.4　表votecontext信息

字 段 名 称	数 据 类 型	字 段 说 明
votecontext_id	int(11)	编号
context	varchar(50)	投票选项内容
Count	int(11)	投票的票数
vote_id	int(11)	编号

在数据库 vote 中 4 张表的关系如图 25.17 所示。

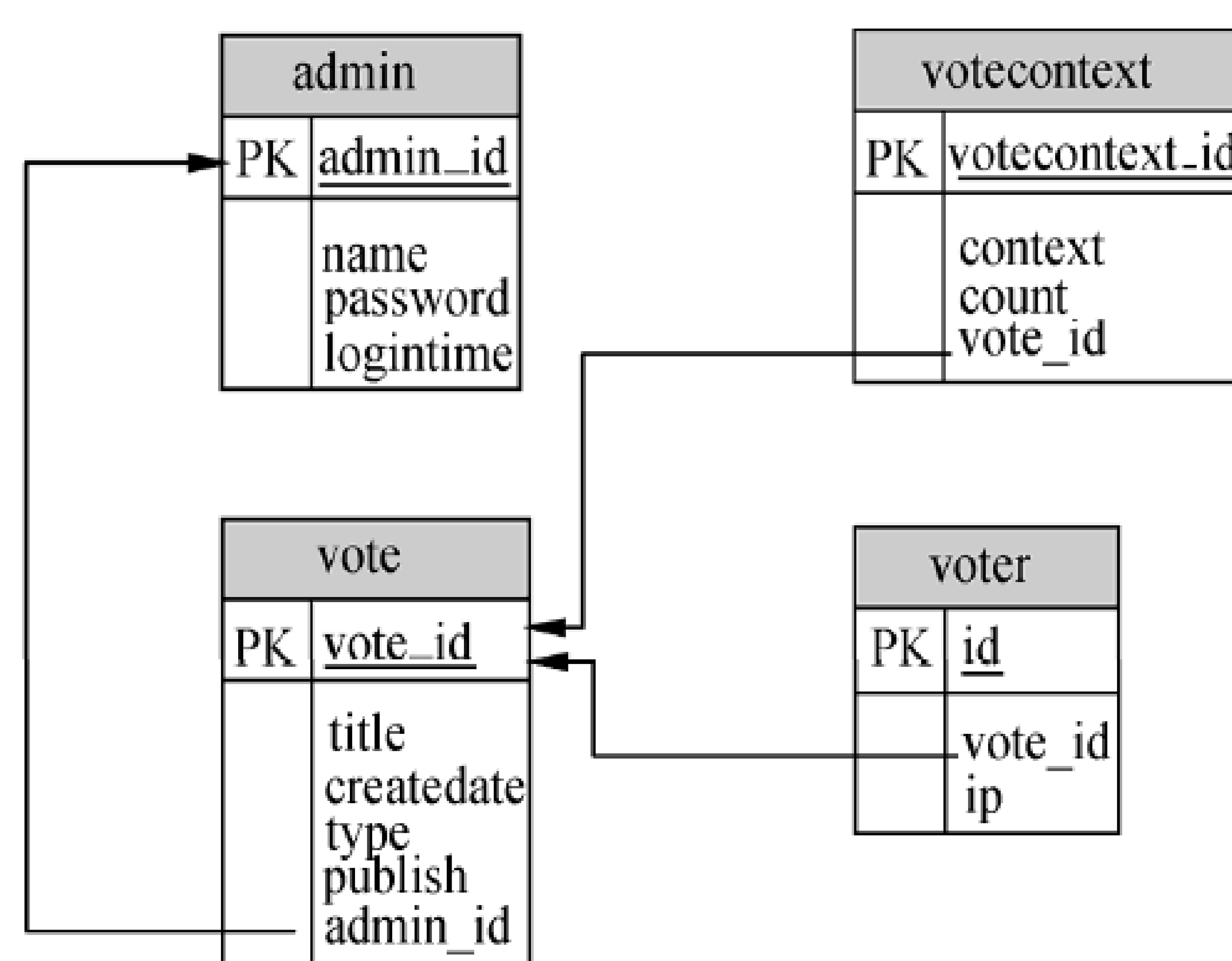


图 25.17　数据库表格的关系

至此，就完成了对投票管理系统数据库和表的设计。

25.2.2　关于 Struts 2.x 的准备

在实现 Struts 2.x 框架、Spring 框架与 Hibernate 三者集成时，对于其中的 Struts 2.0 框架除了需要引入相应的 jar 包外，还必须得对 struts.xml 和 web.xml 文件做相应的配置。

1．引入jar包

首先引入关于 Struts 2.0 框架的核心包：struts2-core-2.0.11.jar、xwork-2.0.4.jar、ognl-2.6.11.jar、freemarker-2.3.8.jar 和 commons-logging-1.0.4.jar。然后由于该框架要与 Spring 框架整合，所以还需要 struts2-spring-plugin-2.0.8.jar。最后由于需要连接数据库 MySQL，

所以还需要引入关于该数据库的驱动 mysql-connector-java-3.1.14-bin.jar。

注意：前 6 个 jar 包在 Struts 2.0 框架的 jar 包中就可以找到，而最后一个关于数据库的驱动则必须从该数据库的官方网站上下载。

2. 修改web.xml文件

为了使投票管理系统项目支持 Struts 2.0 框架，需要在 web.xml 文件中增加如代码 25.1 所示的内容。

代码 25.1　修改 web.xml 文件：web.xml

```
<!--实现对 Spring 框架的监听-->
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
<!--设置过滤器类-->
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>
        org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
</filter>
<!--设置过滤器映射-->
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3. 创建struts.xml文件

为了使投票管理系统项目支持 Struts 2.0 框架，需要在 votesystem/src 目录下创建 struts.xml 文件，该文件的内容如代码 25.2 所示。

代码 25.2　实现 struts.xml 文件：struts.xml

```
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE struts PUBLIC
       "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
       "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <include file="struts-default.xml" />
    <!--设置编码格式-->
    <constant name="struts.i18n.encoding" value="GBK" />
    <!--设置国际化资源-->
    <constant name="struts.custom.i18n.resources" value="globalMessa-
    ges"/>
    <package name="default" extends="struts-default">
...
    </package>
</struts>
```

至此，就完成了对投票管理系统中 Struts 2.0 框架的配置。

25.2.3　关于 Hibernate 3.0 的准备

在实现 Struts 2.x 框架、Spring 框架与 Hibernate 三者集成时，对于其中的 Hibernate 3.0 框架除了需要引入相应的 jar 包外，还必须得对 hibernate.cfg.xml 文件做相应的配置。

1．引入jar文件

首先引入关于 Hibernate 3.0 框架的核心包：Hibernate3.0.jar、log4j-1.2.13.jar、cglib-nodep-2.1_3.jar、dom4j-1.6.1.jar、commons-collections.jar、c3p0-0.9.0.4.jar、jta.jar、antlr-2.7.6.jar。

2．创建hibernate.cfg.xml文件

首先通过 MyEclipse 开发环境中关于 Hibernate 框架的向导，让投票管理系统支持 Hibernate 3.0 框架，然后通过该开发环境的反向工程向导，实现对数据库 vote 中 4 张表进行关系数据库到对象的映射。关于这些持久化类和映射文件可以在具体小节查看，它们分别为 Admin.java、Vote.java、Votecontext.java、Voter.java 和 Admin.hbm.xml、Vote.hbm.xml、Votecontext.hbm.xml、Voter.hbm.xml。hibernate.cfg.xml 文件的内容如代码 25.3 所示。

代码 25.3　实现 hibernate.cfg.xml 文件：hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0
        .dtd">
<hibernate-configuration>

    <session-factory>
        <!-- 指定连接数据库用户名-->
        <property name="connection.username">root</property>
        <!-- 指定连接数据库 URL -->
        <property name="connection.url">
            votesystem:mysql://hostname/votesys
        </property>
        <!-- 指定连接数据库方言-->
        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>
        <!-- 指定连接数据库用户名-->
        <property name="myeclipse.connection.profile">mysql</property>
        <!-- 指定连接数据库密码-->
        <property name="connection.password">root</property>
        <!-- 指定连接数据库驱动-->
        <property name="connection.driver_class">
            com.mysql.votesystem.Driver
        </property>
        <!-- 指定 Hibernate 映射文件 -->
        <mapping resource="./Admin.hbm.xml" />
        <mapping resource="./Vote.hbm.xml" />
```

```
        <mapping resource="./Votecontext.hbm.xml" />
        <mapping resource="./Voter.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

至此，就完成了对投票管理系统中 Hibernate 3.0 框架的配置。

25.2.4　关于 Spring 2.0 的准备

在实现 Struts 2.x 框架、Spring 框架与 Hibernate 三者集成时，对于其中的 Spring 2.0 框架除了需要引入相应的 jar 包外，还必须得对 applicationContext.xml 文件做相应的配置。

1．引入jar文件

首先引入关于 Spring 2.0 框架的核心包：spring.jar。

2．创建applicationContext.xml文件

首先通过 MyEclipse 开发环境中关于 Spring 框架的向导让，投票管理系统支持 Spring 2.0 框架，由于 hibernate.cfg.xml 文件可以合并到 applicationContext.xml 文件中，所以在该系统中删除 hibernate.cfg.xml 文件，然后修改 applicationContext.xml 文件。最后 applicationContext.xml 文件的内容如代码 25.4 所示。

代码 25.4　修改 applicationContext.xml 文件：applicationContext.xml

```
<?xml version="1.0" encoding="GBK"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/beans http:
   //www.springframework.org/schema/beans/spring-beans.xsd">
    <!-- 定义 c3p0 数据源 -->
    <bean id="dataSource"
        class="com.mchange.v2.c3p0.ComboPooledDataSource"
        destroy-method="close">
        <!-- 指定连接数据库驱动 -->
        <property name="driverClass" value="com.mysql.votesystem.Driver" />
        <!-- 指定连接数据库 URL -->
        <property name="votesystemUrl" value="votesystem:mysql://localho-
        st/vote" />
        <!-- 指定连接数据库用户名-->
        <property name="user" value="root" />
        <!-- 指定连接数据库密码-->
        <property name="password" value="root" />
    </bean>
    <!-- 定义 Hibernate 的 sessionFactory -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactory-
        Bean">
        <!-- 指定数据源 -->
        <property name="dataSource" ref="dataSource" />
        <!-- 指定 Hibernate 映射文件 -->
        <property name="mappingResources">
            <list>
                <value>Admin.hbm.xml</value>
```

```
                <value>Vote.hbm.xml</value>
                <value>Voter.hbm.xml</value>
                <value>Votecontext.hbm.xml</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <!-- 指定使用方言 -->
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
                <!-- 是否在控制台输出 SQL 语句 -->
                <prop key="show sql">true</prop>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.votesystem.batch size">20</prop>
            </props>
        </property>
    </bean>
</beans>
```

至此，就完成了对投票管理系统中 Spring 2.0 框架的配置。

25.3　投票管理系统的具体实现——领域模型层

为了让读者可以快速地理解和掌握投票管理系统，在具体讲解时按照面向应用的方式对该系统分成 4 层：领域模型层、业务层、持久层和表示层。本节将详细介绍如何设计领域模型层。

25.3.1　由反向工程生成的领域模型对象

对数据库 vote 中的 4 张表格进行反向工程后，就会自动生成各个表格的对应领域模型对象和映射文件。这些模型的详细描述如表 25.5 所示，各个对象模型的映射文件如表 25.6 所示。

表 25.5　领域模型对象

名　　称	说　　明
Admin.java	管理员模型
Vote.java	投票主题模型
Voter.java	投票人模型
Votecontext.java	投票内容模型

表 25.6　领域模型对象映射

名　　称	映 射 文 件
Admin.java	管理员映射文件
Vote.java	投票主题映射文件
Voter.java	投票人映射文件
Votecontext.java	投票内容映射文件

实现管理员模型的内容如代码 25.5 所示，其映射文件内容如代码 25.6 所示。

代码 25.5　管理员对象：Admin.java

```
...
public class Admin implements java.io.Serializable {
    //创建对应字段的属性
    private Integer adminId;
    private String name;
    private String password;
    private String logintime;
    public Admin() {                                //无参构造函数
    }
    public Admin(String name, String password, String logintime) {
                                                    //有参构造函数
        this.name = name;
        this.password = password;
        this.logintime = logintime;
    }
    //省略属性 adminId、name、password 和 logintime 的 get()和 set()方法
...
}
```

代码 25.6　管理员对象映射文件：Admin.hbm.xml

```
...
<hibernate-mapping>
   <!--制定要持久化的类与对应数据库的表映射关系-->
   <class name="com.cjg.domain.Admin" table="admin" catalog="vote">
      <!--表 admin 主键的设置>
      <id name="adminId" type="java.lang.Integer">
         <column name="admin id" />
         <generator class="native" />
      </id>
      <!--表 admin 字段 name 与属性 name 的映射关系>
      <property name="name" type="java.lang.String">
         <column name="name" length="50" />
      </property>
      <!--表 admin 字段 password 与属性 password 的映射关系>
      <property name="password" type="java.lang.String">
         <column name="password" length="50" />
      </property>
      <!--表 admin 字段 logintime 与属性 logintime 的映射关系>
      <property name="logintime" type="java.lang.String">
         <column name="logintime" length="50" />
      </property>
   </class>
</hibernate-mapping>
```

注意：Admin.java 类可以参考数据库中的表格 admin 来编写，而该类的映射文件则可以通过向导实现。

实现投票主题模型的内容如代码 25.7 所示，其映射文件代码如代码 25.8 所示。

代码 25.7　投票主题对象：Vote.java

```
...
public class Vote implements java.io.Serializable {
    //创建对应字段的属性
    private Integer voteId;
    private String title;
    private String createdate;
    private Integer type;
    private Integer publish;
    private Integer adminId;
    public Vote() {                                    //无参构造函数
    }
    public Vote(String title, String createdate, Integer type, Integer
    publish,                                           //有参构造函数
            Integer adminId) {
        this.title = title;
        this.createdate = createdate;
        this.type = type;
        this.publish = publish;
        this.adminId = adminId;
    }
    //省略 6 个属性的 get()和 set()方法
...
}
```

代码 25.8　投票主题对象映射文件：Vote.hbm.xml

```
...
    <hibernate-mapping>
    <!--制定要持久化的类与对应数据库的表映射关系-->
    <class name="com.cjg.domain.Vote" table="vote" catalog="vote">
        <!--表 vote 主键的设置>
        <id name="voteId" type="java.lang.Integer">
            <column name="vote_id" />
            <generator class="native" />
        </id>
        <!--表 vote 字段 title 与属性 title 的映射关系>
        <property name="title" type="java.lang.String">
            <column name="title" length="50" />
        </property>
        <!--表 vote 字段 createdate 与属性 createdate 的映射关系>
        <property name="createdate" type="java.lang.String">
            <column name="createdate" length="50" />
        </property>
        <!--表 vote 字段 type 与属性 type 的映射关系>
        <property name="type" type="java.lang.Integer">
            <column name="type" />
        </property>
        <!--表 vote 字段 publish 与属性 publish 的映射关系>
        <property name="publish" type="java.lang.Integer">
            <column name="publish" />
        </property>
        <!--表 vote 字段 admin_id 与属性 adminId 的映射关系>
        <property name="adminId" type="java.lang.Integer">
            <column name="admin_id" />
```

```
        </property>
    </class>
</hibernate-mapping>
```

🔔注意：vote.java 类可以参考数据库中的表格 vote 来编写，而该类的映射文件则可以通过向导实现。

实现投票人模型的内容如代码 25.9 所示，其映射文件内容如代码 25.10 所示。

代码 25.9　投票人对象：Voter.java

```
...
public class Voter implements java.io.Serializable {
    //创建对应字段的属性
    private Integer id;
    private Integer voteId;
    private String ip;
    public Voter() {                                    //无参构造函数
    }
    public Voter(Integer voteId, String ip) {           //有参构造函数
        this.voteId = voteId;
        this.ip = ip;
    }
    //省略 3 个属性的 get()和 set()方法
...
}
```

代码 25.10　投票人对象映射文件：Voter.hbm.xml

```
...
<hibernate-mapping>
    <!--制定要持久化的类与对应数据库的表映射关系-->
<class name="com.cjg.domain.Voter" table="voter" catalog="vote">
        <!--表 voter 主键的设置>
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="native" />
        </id>
        <!--表 voter 字段 vote id 与属性 voteId 的映射关系>
        <property name="voteId" type="java.lang.Integer">
            <column name="vote id" />
        </property>
        <!--表 voter 字段 ip 与属性 ip 的映射关系>
        <property name="ip" type="java.lang.String">
            <column name="ip" length="45" />
        </property>
    </class>
</hibernate-mapping>
```

🔔注意：voter.java 类可以参考数据库中的表格 voter 来编写，而该类的映射文件则可以通过向导实现。

实现投票内容模型的内容如代码 25.11 所示，其映射文件内容如代码 25.12 所示。

代码 25.11　投票内容对象：Votecontext.java

```
...
public class Votecontext implements java.io.Serializable {
    //创建对应字段的属性
    private Integer votecontextId;
    private String context;
    private Integer count;
    private Integer voteId;
    public Votecontext() {                          //无参构造函数
    }
    public Votecontext(String context, Integer count, Integer voteId) {
                                                    //有参构造函数
        this.context = context;
        this.count = count;
        this.voteId = voteId;
    }
    //省略 3 个属性的 get()和 set()方法
...
}
```

代码 25.12　投票内容对象映射文件：Votecontext.hbm.xml

```
...
<hibernate-mapping>
   <!--制定要持久化的类与对应数据库的表映射关系-->
   <class name="com.cjg.domain.Votecontext" table="votecontext" catalog=
   "vote">
      <!--表 votecontext 主键的设置>
      <id name="votecontextId" type="java.lang.Integer">
         <column name="votecontext_id" />
         <generator class="native" />
      </id>
      <!--表 votecontext 字段 context 与属性 context 的映射关系>
      <property name="context" type="java.lang.String">
         <column name="context" length="50" />
      </property>
      <!--表 votecontext 字段 count 与属性 count 的映射关系>
      <property name="count" type="java.lang.Integer">
         <column name="count" />
      </property>
      <!--表 votecontext 字段 vote id 与属性 voteId 的映射关系>
      <property name="voteId" type="java.lang.Integer">
         <column name="vote_id" />
      </property>
   </class>
</hibernate-mapping>
```

注意：votecontext.java 类可以参考数据库中的表格 votecontext 来编写，而该类的映射文件则可以通过向导实现。

从上述的 4 段代码中可以发现，这些模型对象（POJO）具有如下规律。

- POJO 类符合 JavaBean 的规范，包含与数据库表中每个字段对应的属性。
- POJO 类一般都有一个整数类型的 ID 属性，用来标识对象，该属性被称为对象标识符（OID，Object Identifier）。
- POJO 类一般都具有一个无参数的构造函数。

25.3.2　程序员设计领域对象

除了数据库 vote 中 4 张表格相对应的领域模型对象外，还需要 3 个领域模型对象：VoteInfo.java、VotingInfo.java 和 Count.java。这些领域模型对象的详细描述如表 25.7 所示。

表 25.7　领域模型对象

名　称	描　述
VoteInfo.java	投票信息模型
VotingInfo.java	投票结果模型
Count.java	投票总数模型

实现投票信息模型的内容如代码 25.13 所示。

代码 25.13　投票信息对象：VoteInfo.java

```
...
public class VoteInfo implements java.io.Serializable{
    private Integer voteId;                       //创建主键属性
    private String title;                         //投票主题属性
    private String createdate;                    //投票创建时间属性
    private String publish;                       //是否开放标识属性
    private String type;                          //投票类型属性
    private String adminname;                     //投票创建人用户名属性
    //省略 6 个属性的 get()和 set()方法
...
}
```

实现投票结果模型的内容如代码 25.14 所示。

代码 25.14　投票结果对象：VotingInfo.java

```
...
public class VotingInfo implements java.io.Serializable{
    private String context;                       //投票子选项内容属性
    private Integer count;                        //投票数属性
    private String percent;                       //百分比属性
    //省略 3 个属性的 get()和 set()方法
...
}
```

实现投票总数模型的内容如代码 25.15 所示。

代码 25.15　投票总数对象：Count.java

```
...
public class Count implements java.io.Serializable{
    private Long singleCount;                     //单选投票主题个数属性
    private Long multiCount;                      //多选投票主题个数属性
    private Long allCount;                        //所有投票主题个数属性
```

```
    //省略 3 个属性的 get()和 set()方法
...
}
```

至此，就完成了对领域模型层的设计。

25.4　投票管理系统的具体实现——持久层

为了让读者可以快速地理解和掌握投票管理系统，在具体讲解时按照面向应用的方式对该系统分成 4 层：领域模型层、业务层、持久层和表示层。本节将详细介绍如何设计持久层。

投票管理系统的持久层采用 DAO 模式设计，由于该系统中使用 Hibernate 3.0 框架，所以在具体实现持久层中的各个类时都继承了 HibernateDaoSupport 类。由于继承了 HibernateDaoSupport 类，所以在实现操作数据库功能时采用 HQL 语言编写。

25.4.1　Admin 模型对象对应的持久层

对于 Admin 模型对象，在持久层中主要用来实现操作该对象的功能。由于该层采用 DAO 模式来设计，所以创建了两个类：AdminDao.java 和 AdminDaoImpl.java。

1．编写操作Admin模型对象接口类

AdminDao.java 文件主要用来定义操作 Admin 模型对象的方法，具体内容如代码 25.16 所示。

代码 25.16　操作 Admin 对象接口类：AdminDao.java

```
...
public interface AdminDao {

    public List<Admin> findAll();            //查找所有 Admin 信息方法
    public void addAdmin(Admin admin);       //增加管理员方法
    public Admin findAdmin(String name, String password);
                                             //通过用户名，密码查找管理员方法
    public void changepwd(Admin admin);      //修改密码方法
    public void updatelogintime(Admin admin);        //修改登录时间方法
    public Admin findNameById(Integer adminId);      //通过 Id 查找管理员方法
    public Admin findAdminByName(String name);
                                             //通过管理员用户名查找管理员方法
}
```

【代码解析】

- findAll()方法用来实现查找所有管理员信息。
- addAdmin()方法用来实现添加管理员信息。
- findAllUsers()方法用来实现通过用户名和密码查找管理员。
- changepwd()方法用来实现修改密码功能。

- updatelogintime()方法用来实现更新登录时间功能。
- findNameById()方法用来实现通过 ID 号来查找管理员。
- findAdminByName()方法用来实现通过名字来查找管理员。

2. 编辑继承AdminDao接口的类

AdminDaoImpl.java 类继承了 AdminDao.java 类，实现了操作 Admin 模型对象的各个方法，具体内容如代码 25.17 所示。

代码 25.17　操作 Admin 对象实现类：AdminDaoImpl.java

```
...
public class AdminDaoImpl extends HibernateDaoSupport implements AdminDao
{
    public List<Admin> findAll() {              //编写查找所有 Admin 信息方法
        return (List<Admin>) getHibernateTemplate().find("from Admin");
    }
    public void addAdmin(Admin admin) {         //编写增加管理员方法
        getHibernateTemplate().save(admin);
    }
    //编写通过用户名、密码查找管理员方法
    public Admin findAdmin(String name, String password) {
        String[] param = new String[] { name, password };
        String sql = "from Admin as admin where admin.name=? and admin.
        password=?";
        List<Admin> list = getHibernateTemplate().find(sql, param);
        if (list != null && list.size() > 0)
            return list.get(0);
        else
            return null;
    }
    public void changepwd(Admin admin) {        //编写修改密码方法
        getHibernateTemplate().update("password", admin);
    }
    public void updatelogintime(Admin admin) { //编写修改登录时间方法
        getHibernateTemplate().update("logintime", admin);
    }
    public Admin findNameById(Integer adminId) {
                                                //编写通过 ID 查找管理员方法
        String sql = "from Admin as admin where admin.adminId=?";
        List<Admin> list = getHibernateTemplate().find(sql, adminId);
        return list.get(0);
    }
    //编写通过管理员用户名查找管理员方法
    public Admin findAdminByName(String name) {
        String sql = "from Admin as admin where admin.name=?";
        List<Admin> list = getHibernateTemplate().find(sql, name);
        if (list != null && list.size() > 0)
            return list.get(0);
        else
            return null;
    }
}
```

注意：在上述代码中，分别实现了 AdminDao 接口中的所有方法。

3．在applicationContext.xml文件中配置DAO

对于关于操作 Admin 模型对象的类，由于要在其他文件中使用，所以必须在applicationContext.xml 文件中对这些类实现依赖注入。下面的代码实现了对操作 Admin 模型对象类的注入。

```
<!--对 AdminDaoImpl 类进行配置 -->
<bean id="adminDao" class="com.cjg.dao.impl.AdminDaoImpl">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

25.4.2　Vote 模型对象对应的持久层

对于 Vote 模型对象，在持久层中主要用来实现操作该对象的功能。由于该层采用 DAO 模式来设计，所以创建了两个类：VoteDao.java 和 VoteDaoImpl.java。

1．编写操作Vote模型对象接口类

VoteDao.java 文件主要用来定义操作 Vote 模型对象的方法，具体内容如代码 25.18 所示。

代码 25.18　操作 Vote 对象接口类：VoteDao.java

```
...
public interface VoteDao {
    public void addVote(Vote vote);                //增加投票主题
    public Integer findIdByTitle(Vote vote);       //通过投票主题查找投票编号
    public List<Vote> findVote();                  //查找所有投票主题
    public Vote findVoteById(Integer voteId);      //通过投票编号查找投票主题
    public void updateVote(Vote vote);             //修改投票信息
    public List<Vote> findVoteByTitle(Vote vote);  //通过投票主题查找投票
    public Long findVoteCountByType(Integer type); //通过投票类型查找投票数
    public Long findVoteCount();                   //查找投票个数
}
```

【代码解析】

- addVote()方法用来实现添加投票主题。
- findIdByTitle()方法用来实现通过投票主题查找投票编号功能。
- findVote()方法用来获取所有的投票主题。
- findVoteById()方法用来通过投票 ID 号来查找投票主题。
- updateVote()方法用来实现更新投票主题。
- findVoteByTitle()方法用来实现通过投票标题来查找投票主题。
- findVoteCountByType()方法用来实现通过投票类型来查找投票内容。
- findVoteCount()方法用来实现查询投票的个数。

2．编辑继承VoteDao接口的类

VoteDaoImpl.java 类继承了 VoteDao.java 类，实现了操作 Vote 模型对象的各个方法，具体内容如代码 25.19 所示。

代码 25.19　操作 Vote 对象实现类：VoteDaoImpl.java

```
...
public class VoteDaoImpl extends HibernateDaoSupport implements VoteDao {
    public void addVote(Vote vote) {        //编写增加投票主题
        getHibernateTemplate().save(vote);
    }
    public Integer findIdByTitle(Vote vote) {  //编写通过投票主题查找投票编号
        String title = vote.getTitle();
        String sql = "from Vote as vote where vote.title=?";
        List<Vote> list = getHibernateTemplate().find(sql, title);
        return list.get(0).getVoteId();
    }
    public List<Vote> findVote() {              //编写查找所有投票主题
        List<Vote> list = getHibernateTemplate().find("from Vote");
        return list;
    }
    public Vote findVoteById(Integer voteId) { //编写通过投票编号查找投票主题
        String sql = "from Vote as vote where vote.voteId=?";
        List<Vote> list = getHibernateTemplate().find(sql, voteId);
        return list.get(0);
    }
    public void updateVote(Vote vote) {         //编写修改投票信息
        getHibernateTemplate().saveOrUpdate(vote);
    }
    public List<Vote> findVoteByTitle(Vote vote) { //编写主题查找投票
        String title = vote.getTitle();
        String sql = "from Vote as vote where vote.title like '%" + title
                + "%'";
        List<Vote> list = getHibernateTemplate().find(sql);
        if (list.size() > 0 && list != null) {
            return list;
        } else
            return null;
    }
    public Long findVoteCountByType(Integer type) {
                                                //编写通过投票类型查找投票数
        String sql = "select count(*) from Vote as vote where vote.type=?";
        List list = getHibernateTemplate().find(sql, type);
        if (list != null && list.size() > 0) {
            return (Long) list.get(0);
        } else
            return new Long(0);
    }
    public Long findVoteCount() {               //编写投票个数
        String sql = "select count(*) from Vote";
        List list = getHibernateTemplate().find(sql);
        if (list != null && list.size() > 0) {
            return (Long) list.get(0);
```

```
        } else
            return new Long(0);
    }
}
```

注意：在上述代码中，分别实现了 VoteDao 接口中的所有方法。

3. 在applicationContext.xml文件配置DAO

对于关于操作 Vote 模型对象的类，由于要在其他文件中使用，所以必须在 applicationContext.xml 文件中对这些类实现依赖注入。下面的代码实现了对操作 Vote 模型对象类的注入。

```
<!--对 VoteDaoImpl 类进行配置 -->
<bean id="voteDao" class="com.cjg.dao.impl.VoteDaoImpl">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

25.4.3　Voter 模型对象对应的持久层

对于 Voter 模型对象，在持久层中主要用来实现操作该对象的功能。由于该层采用 DAO 模式来设计，所以创建了两个类：VoterDao.java 和 VoterDaoImpl.java。

1. 编写操作Vote模型对象接口类

VoterDao.java 文件主要用来定义操作 Voter 模型对象的方法，具体内容如代码 25.20 所示。

代码 25.20　操作 Voter 对象接口类：VoterDao.java

```
...
public interface VoterDao {
    public void addVoter(Voter voter);      //增加投票人信息
    public Voter findVoterByIp(String ip,Integer voteId);
                                            //通过 IP、投票编号查找投票人
}
```

【代码解析】

- addVoter()方法用来实现添加投票人信息。
- findVoterByIp()方法用来实现通过 IP 号查找投票人的功能。

2. 编辑继承VoterDao接口的类

VoterDaoImpl.java 类继承了 VoterDao.java 类，实现了操作 Voter 模型对象的各个方法，具体内容如代码 25.21 所示。

代码 25.21　Voter 对象实现类：VoterDaoImpl.java

```
...
public class VoterDaoImpl extends HibernateDaoSupport implements VoterDao
```

```
{
    public void addVoter(Voter voter) {    //编写投票人信息
        getHibernateTemplate().save(voter);
    }
    public Voter findVoterByIp(String ip, Integer voteId) {
                                            //编写通过 IP、投票编号查找投票人
        Object param[] = { ip, new Integer(voteId) };
        String sql = "from Voter as voter where voter.ip=? and voter.
        voteId=?";
        List<Voter> list = getHibernateTemplate().find(sql, param);
        if (list != null && list.size() > 0) {
            return list.get(0);
        } else {
            return null;
        }
    }
}
```

注意：在上述代码中，分别实现了 VoterDao 接口中的所有方法。

3. 在applicationContext.xml文件中配置DAO

对于关于操作 Votre 模型对象的类，由于要在其他文件中使用，所以必须在 applicationContext.xml 文件中对这些类实现依赖注入。下面的代码实现了对操作 Voter 模型对象类的注入。

```
<!--对 VoterDaoImpl 类进行配置 -->
<bean id="voterDao" class="com.cjg.dao.impl.VoterDaoImpl">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

25.4.4　VoteContext 模型对象对应的持久层

对于 VoteContext 模型对象，在持久层中主要用来实现操作该对象的功能。由于该层采用 DAO 模式来设计，所以创建了两个类：VoteContextDao.java 和 VoteContextDaoImpl.java。

1. 编写操作VoteContext模型对象接口类

VoteContextDao.java 文件主要用来定义操作 VoteContext 模型对象的方法，具体内容如代码 25.22 所示。

代码 25.22　操作 VoteContext 对象接口类：VoteContextDao.java

```
...
package com.cjg.dao;
public interface VoteContextDao {
    public void addVoteContext(Votecontext voteContext);
                                    //添加投票选项信息
    public List<Votecontext> findVoteContextByVoteId(Vote vote);
                                    //通过投票编号查找投票选项
    public void delVoteContext(Votecontext voteContext);
```

```
                                    //删除指定的一条投票选项信息
    public void addOneVoteContext(Votecontext voteContext);
                                    //增加一条投票选项信息
    public void updateVoteContext(Votecontext voteContext);
                                    //更新投票选项信息
                                    //通过投票选项编号查找投票选项信息
    public Votecontext findVCCountByVCId(Votecontext voteContext);
    public Long findTotalCountByVoteId(Integer voteId);
                                    //通过投票编号查找对应该编号的所有投票数
}
```

【代码解析】

- addVoteContext()方法用来实现添加投票选项功能。
- findVoteContextByVoteId()方法通过投票选项 ID 来查找投票选项。
- delVoteContext()方法用来实现删除投票选项。
- addOneVoteContext()方法用来实现添加投票选项。
- updateVoteContext()方法用来实现更新投票选项。
- findVCCountByVCId()方法用来实现通过投票选项编号查找投票选项信息。
- findTotalCountByVoteId()方法用来实现通过投票编号查找对应该编号所有投票数。

2. 编辑继承VoteContextDao接口的类

VoteContextDaoImpl.java 类继承了 VoteContextDao.java 类，实现了操作 VoteContext 模型对象的各个方法，具体内容如代码 25.23 所示。

代码 25.23　操作 VoteContext 对象实现类：VoteContextDaoImpl.java

```
...
public class VotecontextDaoImpl extends HibernateDaoSupport implements
VoteContextDao {
    //编写添加投票选项信息的方法
    public void addVoteContext(Votecontext voteContext) {
        getHibernateTemplate().save(voteContext);
    }
    //编写通过投票编号查找投票选项方法
    public List findVoteContextByVoteId(Vote vote) {
        Integer vote id = vote.getVoteId();
        String sql = "from Votecontext as voteContext where voteContext.
        voteId=?";
        List<Votecontext> list = getHibernateTemplate().find(sql, vote id);
        return list;
    }
    //编写删除指定的一条投票选项信息方法
    public void delVoteContext(Votecontext voteContext) {
        getHibernateTemplate().delete(voteContext);
    }
    //编写增加一条投票选项信息方法
    public void addOneVoteContext(Votecontext voteContext) {
        getHibernateTemplate().save(voteContext);
    }
    //编写更新投票选项信息方法
    public void updateVoteContext(Votecontext voteContext) {
        getHibernateTemplate().saveOrUpdate("voteContextId", voteContext);
    }
```

```
    //编写通过投票选项编号查找投票选项信息方法
    public Votecontext findVCCountByVCId(Votecontext voteContext) {
        Integer vcId = voteContext.getVotecontextId();
        String sql = "from Votecontext as voteContext where voteContext.
        votecontextId=?";
        List<Votecontext> list = getHibernateTemplate().find(sql, vcId);
        return list.get(0);
    }
    //编写通过投票编号查找对应该编号的所有投票数方法
    public Long findTotalCountByVoteId(Integer voteId) {
        String sql = "select sum(voteContext.count) from Votecontext as
        voteContext where voteContext.voteId=?";
        List list = getHibernateTemplate().find(sql, voteId);
        return (Long) list.get(0);
    }
}
```

注意：在上述代码中，分别实现了 VoteContextDao 接口中的所有方法。

3. 在applicationContext.xml文件配置DAO

对于关于操作 VoteContext 模型对象的类，由于要在其他文件中使用，所以必须在 applicationContext.xml 文件中对这些类实现依赖注入。下面的代码实现了对操作 VoteContext 模型对象类的注入。

```
<!--对 VotecontextDaoImpl 类进行配置 -->
<bean id="voteContextDao"
    class="com.cjg.dao.impl.VotecontextDaoImpl">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

至此，就完成了对持久层的设计。

25.5　投票管理系统的具体实现——业务层

为了让读者可以快速地理解和掌握投票管理系统，在具体讲解时按照面向应用的方式对该系统分成 4 层：领域模型层、业务层、持久层和表示层。本节将详细介绍如何设计业务层。

投票管理系统的业务层采用 DAO 模式设计，由于该系统中使用 Sping 2.0 框架来解决该层与其他层的耦合问题，所以在实现业务层的各个类都需要在 applicationContext.xml 文件中实现依赖注入。

25.5.1　Admin 模型对象对应的业务层

对于 Admin 模型对象，在业务层中主要用来实现与持久层的交互处理和事务管理。由于该层采用 DAO 模式来设计，所以创建了两个类：AdminService.java 和 AdminServiceImpl.java。

1．业务层关于Admin模型对象接口类

AdminService.java 文件主要用来定义与持久层相互交换的方法，具体内容如代码 25.24 所示。

代码 25.24　操作 Admin 对象接口类：AdminService.java

```
...
public interface AdminService {
    public List<Admin> findAll();               //查找所有 Admin 信息
    public void addAdmin(Admin admin);          //增加管理员
    public Admin findAdmin(String name, String password);
                                                //通过用户名，密码查找管理员
    public void changepwd(Admin admin);         //修改密码
    public void updatelogintime(Admin admin);   //修改登录时间
    public Admin findNameById(Integer adminId);//通过 ID 查找管理员
    public Admin findAdminByName(String name); //通过管理员用户名查找管理员
}
```

【代码解析】

- findAll()方法用来实现查找所有管理员信息。
- addAdmin()方法用来实现添加管理员信息。
- findAllUsers()方法用来实现通过用户名和密码查找管理员。
- changepwd()方法用来实现修改密码功能。
- updatelogintime()方法用来实现更新登录时间功能。
- findNameById()方法用来实现通过 ID 号来查找管理员。
- findAdminByName()方法用来实现通过名字来查找管理员。

2．编辑继承AdminService接口的类

AdminServiceImpl.java 类继承了 AdminService.java 类，实现了接口类中定义的各个方法，具体内容如代码 25.25 所示。

代码 25.25　操作 Admin 对象实现类：AdminServiceImpl.java

```
...
public class AdminServiceImpl implements AdminService {
    private AdminDao adminDao;              //创建一个 adminDao 属性

    public AdminDao getAdminDao() {         //配置 adminDao 属性
        return adminDao;
    }
    public void setAdminDao(AdminDao adminDao) {
        this.adminDao = adminDao;
    }
    public List<Admin> findAll() {          //编写查找所有 Admin 信息方法
        List<Admin> list = adminDao.findAll();
        return list;
    }
    public void addAdmin(Admin admin) {     //编写增加管理员方法
        adminDao.addAdmin(admin);
    }
```

```
    //编写通过用户名，密码查找管理员方法
    public Admin findAdmin(String name, String password) {
        return adminDao.findAdmin(name, password);
    }
    public void changepwd(Admin admin) {   //编写修改密码方法
        adminDao.changepwd(admin);
    }
    public void updatelogintime(Admin admin) {     //编写修改登录时间方法
        adminDao.updatelogintime(admin);
    }
    public Admin findNameById(Integer adminId) {//编写通过 ID 查找管理员方法
        return adminDao.findNameById(adminId);
    }
    public Admin findAdminByName(String name) {
                                    //编写通过管理员用户名查找管理员方法
        return adminDao.findAdminByName(name);
    }
}
```

注意：在上述代码中，分别实现了 AdminServiceImpl 接口中的所有方法。

3．在applicationContext.xml文件配置DAO

对于投票管理系统在业务层采用了 Spring 框架，所以需要在 applicationContext.xml 文件中对这些类实现依赖注入。下面的代码实现了对 AdminServiceImpl 类的注入。

```
<!--对 AdminServiceImpl 类进行配置 -->
<bean id="adminService"
    class="com.cjg.service.impl.AdminServiceImpl">
    <property name="adminDao" ref="adminDao" />
</bean>
```

25.5.2　Vote 模型对象对应的业务层

对于 Vote 模型对象，在业务层中主要用来实现与持久层的交互处理和事务管理。由于该层采用 DAO 模式来设计，所以创建了两个类：VoteService.java 和 VoteServiceImpl.java。

1．业务层关于Vote模型对象接口类

VoteService.java 文件主要用来定义与持久层相互交换的方法，具体内容如代码 25.26 所示。

代码 25.26　操作 Vote 对象接口类：VoteService.java

```
...
public interface VoteService {
    public void addVote(Vote vote);             //添加投票信息方法
    public Integer findIdByTitle(Vote vote);    //通过投票主题查找投票 ID 号方法
    public List<Vote> findVote();               //查找所有投票方法
    public Vote findVoteById(Integer voteId);   //通过投票 ID 查找投票
    public void updateVote(Vote vote);          //更新投票
    public List<Vote> findVoteByTitle(Vote vote);
```

```
                                             //通过投票主题查找所有的投票
    public Long findVoteCountByType(Integer type);
                                             //通过投票类型查找投票数
    public Long findVoteCount();             //查找所有的投票数
}
```

【代码解析】

- addVote()方法用来实现添加投票主题。
- findIdByTitle()方法用来实现通过投票主题查找投票编号功能。
- findVote()方法用来获取所有的投票主题。
- findVoteById()方法用来通过投票 ID 号来查找投票主题。
- updateVote()方法用来实现更新投票主题。
- findVoteByTitle()方法用来实现通过投票标题来查找投票主题。
- findVoteCountByType()方法用来实现通过投票类型来查找投票内容。
- findVoteCount()方法用来实现查询投票的个数。

2．编辑继承VoteService接口的类

VoteServiceImpl.java 类继承了 VoteService.java 类，实现了接口类中定义的各个方法，具体内容如代码 25.27 所示。

代码 25.27　操作 Vote 对象实现类：VoteServiceImpl.java

```
...
public class VoteServiceImpl implements VoteService {
    private VoteDao voteDao;             //创建字段 voteDao
    public VoteDao getVoteDao() {        //配置属性 voteDao
        return voteDao;
    }
    public void setVoteDao(VoteDao voteDao) {
        this.voteDao = voteDao;
    }
    public void addVote(Vote vote) {     //实现添加投票方法
        voteDao.addVote(vote);
    }
    public Integer findIdByTitle(Vote vote) {
                                         //实现通过投票主题查找投票 ID 号方法
        return voteDao.findIdByTitle(vote);
    }
    public List<Vote> findVote() {       //实现查找所有投票方法
        return voteDao.findVote();
    }
    public Vote findVoteById(Integer voteId) {
                                         //实现通过投票 ID 查找投票方法
        return voteDao.findVoteById(voteId);
    }
    public void updateVote(Vote vote) {//实现更新投票方法
        voteDao.updateVote(vote);
    }
    public List<Vote> findVoteByTitle(Vote vote) {
                                         //实现通过投票主题查找投票方法
        return voteDao.findVoteByTitle(vote);
    }
```

```
    public Long findVoteCount() {       //实现查找投票总数方法
        return voteDao.findVoteCount();
    }
    public Long findVoteCountByType(Integer type) {
                                        //实现通过投票类型查找投票总数的方法
        return voteDao.findVoteCountByType(type);
    }
}
```

注意：在上述代码中，分别实现了 VoteServiceImpl 接口中的所有方法。

3．在applicationContext.xml文件配置DAO

对于投票管理系统在业务层采用了 Spring 框架，所以需要在 applicationContext.xml 文件中对这些类实现依赖注入。下面的代码实现了对 VoteServiceImpl 类的注入。

```
<!--对 VoteServiceImpl 类进行配置 -->
<bean id="voteService"
    class="com.cjg.service.impl.VoteServiceImpl">
    <property name="voteDao" ref="voteDao" />
</bean>
```

25.5.3　Voter 模型对象对应的业务层

对于 Voter 模型对象，在业务层中主要用来实现与持久层的交互处理和事务管理。由于该层采用 DAO 模式来设计，所以创建了两个类：VoterService.java 和 VoterServiceImpl.java。

1．业务层关于Voter模型对象接口类

VoterService.java 文件主要用来定义与持久层相互交换的方法，具体内容如代码 25.28 所示。

代码 25.28　操作 Voter 对象接口类：VoterService.java

```
...
public interface VoterService {
    public void addVoter(Voter voter); //添加投票人
    public Voter findVoterByIp(String ip,Integer voteId);
                                        //通过投票人的 IP 和投票 ID 查找投票人
}
```

【代码解析】

- addVoter()方法用来实现添加投票人信息。
- findVoterByIp()方法用来实现通过 IP 号查找投票人的功能。

2．编辑继承VoterService接口的类

VoterServiceImpl.java 类继承了 VoterService.java 类，实现了接口类中定义的各个方法，具体内容如代码 25.29 所示。

代码 25.29　操作 Voter 对象实现类：VoterServiceImpl.java

```
...
public class VoterServiceImpl implements VoterService {
    private VoterDao voterDao;                    //创建字段 voterDao

    public VoterDao getVoterDao() {               //配置属性 voterDao
        return voterDao;
    }
    public void setVoterDao(VoterDao voterDao) {
        this.voterDao = voterDao;
    }
    public void addVoter(Voter voter) {           //实现添加投票人方法
        voterDao.addVoter(voter);
    }
    //实现通过投票人的 IP 和投票 ID 查找投票人
    public Voter findVoterByIp(String ip, Integer voteId) {
        return voterDao.findVoterByIp(ip, voteId);
    }
}
```

注意：在上述代码中，分别实现了 VoterService 接口中的所有方法。

3．在applicationContext.xml文件中配置DAO

对于投票管理系统在业务层采用了 Spring 框架，所以需要在 applicationContext.xml 文件中对这些类实现依赖注入。下面的代码实现了对 VoterServiceImpl 类的注入。

```
<!--对 VoteServiceImpl 类进行配置 -->
<bean id="voterService"
    class="com.cjg.service.impl.VoterServiceImpl">
    <property name="voterDao" ref="voterDao" />
</bean>
```

25.5.4　VoteContext 模型对象对应的业务层

对于 VoteContext 模型对象，在业务层中主要用来实现与持久层的交互处理和事务管理。由于该层采用 DAO 模式来设计，所以创建了两个类：VoteContextService.java 和 VoteContextServiceImpl.java。

1．业务层关于VoteContext模型对象接口类

VoteContextService.java 文件主要用来定义与持久层相互交换的方法，具体内容如代码 25.30 所示。

代码 25.30　操作 VoteContext 对象接口类：VoteContextService.java

```
...
public interface VoteContextService {

    public void addVoteContext(Votecontext voteContext);   //添加投票选项
    public List<Votecontext> findVoteContextByVoteId(Vote vote);
```

```
                                            //通过投票 ID 查找投票选项
    public void delVoteContext(Votecontext voteContext);   //删除投票选项
    public void addOneVoteContext(Votecontext voteContext);
                                            //添加某一投票选项
    public void updateVoteContext(Votecontext voteContext);
                                            //更新投票选项
    public Votecontext findVCCountByVCId(Votecontext voteContext);
                                            //查找投票里的投票选项
    public Long findTotalCountByVoteId(Integer voteId);
                                            //查找投票里的投票选项数
}
```

【代码解析】

- addVoteContext()方法用来实现添加投票选项功能。
- findVoteContextByVoteId()方法通过投票选项 ID 来查找投票选项。
- delVoteContext()方法用来实现删除投票选项。
- addOneVoteContext()方法用来实现添加投票选项。
- updateVoteContext()方法用来实现更新投票选项。
- findVCCountByVCId()方法用来实现通过投票选项编号查找投票选项信息。
- findTotalCountByVoteId()方法用来实现通过投票编号查找对应该编号的所有投票数。

2. 编辑继承VoteContextService接口的类

VoteContextServiceImpl.java 类继承了 VoteContextService.java 类，实现了接口类中定义的各个方法，具体内容如代码 25.31 所示。

代码 25.31　操作 VoteContext 对象实现类：VoteContextServiceImpl.java

```
...
public class VoteContextServiceImpl implements VoteContextService {
    VoteContextDao voteContextDao;                  //创建字段 voteContextDao

    public VoteContextDao getVoteContextDao() {//配置属性 voteContextDao
        return voteContextDao;
    }
    public void setVoteContextDao(VoteContextDao voteContextDao) {
        this.voteContextDao = voteContextDao;
    }
    //实现添加投票选项方法
    public void addVoteContext(Votecontext voteContext) {
        voteContextDao.addVoteContext(voteContext);
    }
    //实现通过投票 ID 查找投票选项方法
    public List<Votecontext> findVoteContextByVoteId(Vote vote) {
        return voteContextDao.findVoteContextByVoteId(vote);
    }
    //实现删除投票选项方法
    public void delVoteContext(Votecontext voteContext) {
        voteContextDao.delVoteContext(voteContext);
    }
    //实现添加某一投票选项方法
    public void addOneVoteContext(Votecontext voteContext) {
```

```
        voteContextDao.addOneVoteContext(voteContext);
    }
    //实现更新投票选项方法
    public void updateVoteContext(Votecontext voteContext) {
        voteContextDao.updateVoteContext(voteContext);
    }
    //实现查找投票里的投票选项
    public Votecontext findVCCountByVCId(Votecontext voteContext) {
        return voteContextDao.findVCCountByVCId(voteContext);
    }
    //查找投票里的投票选项数
    public Long findTotalCountByVoteId(Integer voteId) {
        return voteContextDao.findTotalCountByVoteId(voteId);
    }
}
```

注意：在上述代码中，分别实现了 VoteContextService 接口中的所有方法。

3. 在applicationContext.xml文件配置DAO

对于投票管理系统在业务层采用了 Spring 框架，所以需要在 applicationContext.xml 文件中对这些类实现依赖注入。下面的代码实现了对 VoteContextServiceImpl 类的注入。

```
<!--对 VoteContextServiceImpl 类进行配置 -->
<bean id="voteContextService"
    class="com.cjg.service.impl.VoteContextServiceImpl">
    <property name="voteContextDao" ref="voteContextDao" />
</bean>
```

至此，就完成了对业务层的设计。

25.6　关于管理员表示层

为了让读者可以快速地理解和掌握投票管理系统，在具体讲解时按照面向应用的方式对该系统分成 4 层：领域模型层、业务层、持久层和表示层。本节将详细介绍如何设计表示层。

该节主要介绍如何设计管理员操作的表示层，关于管理员操作的所有请求都由 Struts 2.0 框架的 Action 类来处理。由于要利用 Sping 2.0 框架来解决该层与其他层的耦合问题，所以所有的 Action 类都需要在 applicationContext.xml 文件中实现依赖注入。

25.6.1　关于管理员的登录和退出

在设计关于管理员的登录和退出表现层时，利用 Struts 2.0 框架来实现页面的跳转。该操作涉及的页面是实现管理员登录的页面 login.jsp。

关于管理员登录和退出的操作由 Struts 2.0 框架中，名为 Login 和 Logout 的 Action 类来处理，Login.java 的具体内容如代码 25.32 所示。Logout.java 的具体内容如代码 25.33 所示。

代码 25.32　关于管理员登录的 Action：Login.java

```
...
public class Login extends AdminRoot {
    public String execute() throws Exception {
        HttpSession session = ServletActionContext.getRequest().getSes-
        sion();
        if (session.getAttribute("rand") == null) {
            return ERROR;
        }
        String sf = (String) session.getAttribute("rand");//获得图形校验码
        // 如果图形校验码正确，判断账号、密码是否正确
        if (sf.equals(safecode)) {
            Admin admin = adminService.findAdmin(name, password);
            if (admin == null) {
                addActionError(getText("loginerror"));
                return ERROR;
            } else {
                session.setAttribute("admin", admin);
                                                //将 admin 信息存入 Session
                Count count2 = new Count();
                count2.setSingleCount(voteService  //设置单选投票个数
                        .findVoteCountByType(new Integer(1)));
                count2.setMultiCount(voteService   //设置多选投票个数
                        .findVoteCountByType(new Integer(0)));
                // 设置所有投票个数
                count2.setAllCount(voteService.findVoteCount());
                session.setAttribute("count2", count2);
                // 获得当前系统时间并格式化，存入管理员登录时间
                Date date = Calendar.getInstance().getTime();
                SimpleDateFormat formatter = new SimpleDateFormat(
                        "yyyy-MM-dd HH:mm:ss");
                String dateString = formatter.format(date);
                admin.setLogintime(dateString);
                adminService.updatelogintime(admin);//更新管理员登录时间信息
                return SUCCESS;
            }
        } else {
            addActionError(getText("codeerror"));
            return ERROR;
        }
    }
}
```

代码 25.33　关于管理员退出的 Action：Logout.java

```
...
public class Logout extends ActionSupport {
    public String execute() throws Exception {
        //获取 Session 对象
        HttpSession session = ServletActionContext.getRequest().getSes-
        sion();
        session.invalidate();       //调用 invalidate()方法
        return SUCCESS;
    }
}
...
```

首先在 applicationContext.xml 文件中配置 Login 和 Logout 类。

```
<!--对 Login 类进行配置 -->
<bean id="login" class="com.cjg.action.admin.Login">
    <property name="adminService" ref="adminService" />
    <property name="voteService" ref="voteService" />
</bean>
<!--对 Logout 类进行配置 -->
<bean id="logout" class="com.cjg.action.admin.Logout"/>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 login 的 Action -->
<action name="login" class="login">
    <result>/jsp/frame/frame.jsp</result>
    <result name="error">/login.jsp</result>
</action>
<!--配置名为 logout 的 Action -->
<action name="logout" class="logout">
    <result type="redirect">/login.jsp</result>
</action>
```

在配置文件 struts.xml 中涉及的页面是 login.jsp，该页面用来实现管理员登录，其具体内容如代码 25.34 所示。

代码 25.34　管理员登录页面：login.jsp

```
...
    <body>
            <!--输出错误信息-->
            <font color="red"> <s:actionerror /> <s:fielderror /> </font>
            <!--表单-->
            <s:form action="loginValidate" theme="simple">
                <table  background="jsp/img/denglukuang.jpg"  width="344"
                    height="300">
                        <!--用户名输入框-->
                        <td>
                            <s:text name="inputusername" />
                            <s:textfield name="name" />
                        </td>
                        <!--密码输入框-->
                        <td>
                            <s:text name="inputpsw" />
                            <s:password name="password" />
                        </td>
                        <!--验证码输入框-->
                        <td>
                            <s:text name="code" />
                            <s:textfield name="safecode" />
                            <img src="safecode" id="safecode" />
                        </td>
                            <!--提交和返回按钮-->
                            <s:submit value="%{getText('submit')}" />
                            <s:reset value="%{getText('reset')}" />
                        </td>
                </table>
            </s:form>
    </body>
...
```

25.6.2　创建新管理员

在设计关于创建新管理员表现层时，利用 Struts 2.0 框架来实现页面的跳转。该操作涉及的页面是实现创建新管理员的页面 createadmin.jsp。关于创建新管理员的操作由 Struts 2.0 框架中名为 Createadmin 的 Action 类来处理，Createadmin.java 的具体内容如代码 25.35 所示。

代码 25.35　创建新的管理员 Action：Createadmin.java

```
...
public class Createadmin extends AdminRoot {
    private static final String CREATEADMINERROR = "createAdminError";
    public String execute() throws Exception {
        //通过用户名查找管理员信息
        Admin a = adminService.findAdminByName(name);
        if (a != null) {                        // 如果不为空，说明已经存在
            addActionError(getText("adminexist"));
            return CREATEADMINERROR;
        } else {
            //获取时间变量
            Date date = Calendar.getInstance().getTime();
            //创建时间格式
            SimpleDateFormat formatter = new SimpleDateFormat(
                    "yyyy-MM-dd HH:mm:ss");
            String dateString = formatter.format(date);
                                                //获取符合时间格式的时间
            Admin admin = new Admin();          //创建 admin 对象
            admin.setName(name);                //设置名字
            admin.setPassword(newpwd1);         //设置密码
            admin.setLogintime(dateString);//设置时间
            adminService.addAdmin(admin);
            return SUCCESS;
        }
    }
}
```

首先在 applicationContext.xml 文件中配置 Createadmin 类。

```
<!--对 Createadmin 类进行配置 -->
<bean id="createadmin"
    class="com.cjg.action.admin.Createadmin">
    <property name="adminService" ref="adminService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 createadminValidate 的类 -->
    <action name="createadminValidate"
        class="com.cjg.action.validators.CreateAdminValidate">
        <result name="input">/jsp/admin/createadmin.jsp</result>
        <result type="chain">createadmin</result>
        <interceptor-ref name="defaultStack" />
        <interceptor-ref name="SessionInterceptor" />
    </action>
```

在配置文件 struts.xml 中涉及的页面是 createadmin.jsp，该页面用来创建新管理员，其具体内容如代码 25.36 所示。

代码 25.36　创建新管理员页面：createadmin.jsp

```
...
    <body>
...
                <!--输出错误信息 -->
                <font color="red"> <s:fielderror /> <s:actionerror />
                </font>
                <!--表单 -->
                <s:form action="createadminValidate" theme="simple">
                    <table>
                            <!--用户名输入框 -->
                            <td>
                                <s:text name="inputusername" />
                                <s:textfield name="name" />
                            </td>
                            <!--密码输入框 -->
                            <td>
                                <s:text name="inputpsw" />
                                <s:textfield name="newpwd1" />
                            </td>
                            <!--确认密码输入框 -->
                            <td>
                                <s:text name="inputpswagain" />
                                <s:textfield name="newpwd2" />
                            </td>
                            <!--提交和返回按钮 -->
                            <td>
                                <s:submit key="submit" />
                                <s:reset key="reset" />
                            </td>
                    </table>
                </s:form>
    </body>
...
```

25.6.3　更改管理员密码

在设计关于更改管理员密码表现层时，利用 Struts 2.0 框架来实现页面的跳转。该操作涉及的页面是实现更改管理员密码的页面 changepwd.jsp 和更改密码成功页面 changepwd.jsp。关于更改管理员密码的操作，由 Struts 2.0 框架中名为 ChangePwd 的 Action 类来处理，ChangePwd.java 的具体内容如代码 25.37 所示。

代码 25.37　修改管理员密码的 Action：ChangePwd.java

```
...
public class ChangePwd extends AdminRoot {
    private static final String PWDERROR = "pwderror";
    public String execute() throws Exception {
```

```
        HttpSession session = ServletActionContext.getRequest().getSess-
        ion();
        // 从 Session 中获得当前登录管理员信息
        Admin admin = (Admin) session.getAttribute("admin");
        // 判断旧密码是否正确
        if (admin.getPassword().equals(password)) {
            admin.setPassword(newpwd1);
            // 更新管理员密码信息
            adminService.changepwd(admin);
            return SUCCESS;
        } else
            addActionError(getText("oldpswerror"));
        return PWDERROR;
    }
}
```

首先在 applicationContext.xml 文件中配置 ChangePwd 类。

```
<!--对 ChangePwd 类进行配置 -->
<bean id="changepwd" class="com.cjg.action.admin.ChangePwd">
    <property name="adminService" ref="adminService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 changepwd 的 action -->
<action name="changepwd" class="changepwd">
    <result name="success">
        /jsp/admin/changepwdsuccess.jsp
    </result>
    <result name="pwderror">/jsp/admin/changepwd.jsp</result>
    <interceptor-ref name="defaultStack" />
    <interceptor-ref name="SessionInterceptor" />
</action>
```

1. 关于修改管理员密码的页面

页面 changepwd.jsp 用来修改管理员密码，该页面的具体内容如代码 25.38 所示。

代码 25.38　修改管理员密码页面：changepwd.jsp

```
...
    <body>
        <!--显示导航链接 -->
        <a href="../frame/main.jsp"> <s:property
                value="%{getText('mainpage')}" /> </a>
        <a href="adminmanage.jsp"> <s:property
                value="%{getText('adminmanage')}" /> </a>
        <s:property value="%{getText('changepsw')}" />
            <s:property value="%{getText('changeadminpsw')}" />
            <font color="red"> <s:actionerror /> <s:fielderror /> </font>
            <s:form action="changepwdValidate" theme="simple">
                <table>
                        <td>
                            <!--旧密码输入框-->
                            <s:text name="inputoldpsw" />
                            <s:textfield name="password" />
                        </td>
                        <td>
```

```
                    <!--新密码输入框-->
                    <s:text name="inputnewpsw" />
                    <s:textfield name="newpwd1" />
                </td>
                <td>
                    <!--确认密码输入框-->
                    <s:text name="inputnewpswagain" />
                    <s:textfield name="newpwd2" />
                </td>
                <td>
                    <!--提交和返回按钮-->
                    <s:submit key="submit" />
                    <s:reset key="reset" />
                </td>
            </table>
        </s:form>
    </body>
...
```

2. 关于修改管理员密码成功的页面

changepwd.jsp 页面为修改管理员密码成功的页面，该页面的具体内容如代码 25.39 所示。

代码 25.39　修改管理员密码成功页面：changepwdsuccess.jsp

```
...
    <body>
        <!--显示导航链接 -->
        <a href="../frame/frame.html"> <s:property
                value="%{getText('mainpage')}" /> </a>
        <!--获取用户名-->
        <a href="../admin/adminmanage.jsp"> <s:property
                value="%{getText('adminmanage')}" /> </a>
        <!--获取用户密码-->
        <a href="../admin/changepwd.jsp"> <s:property
                value="%{getText('changepsw')}" /> </a>
        <!--获取密码修改成功信息-->
        <center>
            <s:property value="%{getText('changepswsuccess')}" />
        </center>
    </body>
...
```

25.7　关于创建投票表示层

为了让读者可以快速地理解和掌握投票管理系统，在具体讲解时按照面向应用的方式对该系统分成 4 层：领域模型层、业务层、持久层和表示层。本节将详细介绍如何设计表示层。

该节主要介绍如何设计创建投票操作的表示层，关于创建投票操作的所有请求都由 Struts 2.0 框架的 Action 类来处理。由于要利用 Spring 2.0 框架来解决该层与其他层的耦合

问题，所以所有的 Action 类都需要在 applicationContext.xml 文件中实现依赖注入。

25.7.1　创建新的投票主题

在设计关于创建投票主题表现层时，利用 Struts 2.0 框架来实现页面的跳转。该操作涉及的页面有创建新投票主题页面 newvote.jsp。关于创建投票主题的操作由 Struts 2.0 框架中名为 NewVote 的 Action 类来处理，NewVote.java 的具体内容如代码 25.40 所示。

代码 25.40　创建新的投票主题 Action：NewVote.java

```
...
public class NewVote extends VoteRoot {
    private static final String ADDVOTEERROR = "addVoteError";
    public String execute() throws Exception {
        //获取 Session 对象
        HttpSession session = ServletActionContext.getRequest().getSess-
        ion();
        Vote vote = new Vote();                    //创建 vote 对象
        //设置变量 vote 的值
        vote.setTitle(title);
        vote.setType(type);
        vote.setPublish(publish);
        // 获得当前系统时间并格式化，存入创建投票时间
        Date date = Calendar.getInstance().getTime();
        //设置时间格式
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd
        HH:mm:ss");
        String dateString = formatter.format(date);
        vote.setCreatedate(dateString);
        vote.setAdminId(((Admin) session.getAttribute("admin")).getAdmin
        Id());
        List l = voteService.findVoteByTitle(vote);//设置变量 l 的值
        if (l != null) {                           //判断 l 的值
            addActionError(getText("voteexist"));
            return ADDVOTEERROR;
        } else {
            session.setAttribute("vote", vote);
            session.setAttribute("contextcount", contextcount);
            return SUCCESS;
        }
    }
}
```

首先在 applicationContext.xml 文件中配置 NewVote 类。

```
<!--对 NewVote 类进行配置 -->
<bean id="newvote" class="com.cjg.action.vote.NewVote">
    <property name="voteService" ref="voteService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 newvote 的 action -->
<action name="newVote" class="newvote">
    <result name="success">/jsp/vote/newvotecontext.jsp</result>
```

```
    <result name="addVoteError">/jsp/vote/newvote.jsp</result>
    <result name="invalid.token">index.jsp</result>
    <interceptor-ref name="defaultStack" />
    <interceptor-ref name="tokenSession" />
    <interceptor-ref name="SessionInterceptor" />
</action>
```

在配置文件 struts.xml 中涉及的页面是 newvote.jsp，该页面用来创建新投票主题，其具体内容如代码 25.41 所示。

代码 25.41　创建新投票主题页面：newvote.jsp

```
...
    <body>
        <!--显示导航链接 -->
        <a href="../frame/main.jsp"><s:property
                value="%{getText('mainpage')}" /> </a>
        <s:property value="%{getText('createvote')}" />
        <s:property value="%{getText('createnewvote')}" />
        <center>
            <s:property value="%{getText('inputvoteinfo')}" />
            <font color="red"> <s:fielderror /> <s:actionerror /> </font>
            <s:form action="newvoteValidate" theme="simple">
                <table>
                        <tr><td>
                            <!--投票主题->
                            <s:text name="votetitle" />
                            <s:textfield name="title" />
                        </td></tr><tr><td>
                            <!--投票个数->
                            <s:text name="countofvotecontext" />
                            <s:textfield name="contextcount" />
                        </td><td>
                            <!--投票类型选择框-->
                            <s:select list="#{'1':'单选','0':'多选'}" name=
                            "type" />
                        </td></tr><tr><td>
                            <s:text name="publishofvote" />
                        </td><td>
                            <!--投票是否开放多选框-->
                            <s:select list="#{'1':'是','0':'否'}" name=
                            "publish" />
                        </td></tr><tr><td></td><td>
                            <!--提交按钮-->
                            <s:submit key="next" /></td></tr>
                </table>
            </s:form>
        </center>
    </body>
...
```

25.7.2　创建投票选项的投票选项

在设计关于创建投票选项表现层时，利用 Struts 2.0 框架来实现页面的跳转。该操作涉

及的页面是创建投票选项页面 newvotecontext.jsp。关于创建投票主题的操作由 Struts 2.0 框架中名为 NewVoteContext 的 Action 类来处理，NewVoteContext.java 的具体内容如代码 25.42 所示。

代码 25.42　创建投票选项 Action：NewVoteContext.java

```
...
public class NewVoteContext extends VoteContextRoot {
    public String execute() throws Exception {
        HttpSession session = ServletActionContext.getRequest().getSession();
        // 从 Session 中获得投票信息
        Vote vote = (Vote) session.getAttribute("vote");
        // 将投票标题信息添加到数据库
        voteService.addVote(vote);
        Integer vote id = voteService.findIdByTitle(vote);
        // 从页面获得投票选项信息（数组类型）
        String[] voteContext = context;
        Votecontext votecontext;
        for (int i = 0; i < voteContext.length; i++) {
            votecontext = new Votecontext();
            votecontext.setVoteId(vote_id);
            votecontext.setContext(voteContext[i]);
            votecontext.setCount(0);
            // 将投票选项信息添加到数据库
            voteContextService.addVoteContext(votecontext);
        }
        return SUCCESS;
    }
}
```

首先在 applicationContext.xml 文件中配置 NewVoteContext 类。

```
<!--对 NewVoteContext e 类进行配置 -->
<bean id="newVoteContext"
    class="com.cjg.action.vote.NewVoteContext">
    <property name="voteContextService" ref="voteContextService" />
    <property name="voteService" ref="voteService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 newVoteContext 的 action-->
<action name="newVoteContext" class="newVoteContext">
    <result type="chain">showVote</result>
    <interceptor-ref name="defaultStack" />
    <interceptor-ref name="tokenSession" />
    <interceptor-ref name="SessionInterceptor" />
</action>
```

在配置文件 struts.xml 中涉及的页面有 newvotecontext.jsp，该页面用来创建投票选项内容，其具体内容如代码 25.43 所示。

代码 25.43　创建投票选项页面：newvotecontext.jsp

```
...
    <body>
        <!--显示导航链接 -->
```

```
        <a href="../frame/main.jsp"> <s:propertyvalue="%{getText('main-
        page')}" /> </a>
        <s:property value="%{getText('createvote')}" />
        <s:property value="%{getText('inputthecontextofnewvote')}" />
        <!--表单 -->
        <s:form action="newVoteContext" theme="simple">
            <s:bean name="org.apache.struts2.util.Counter" id="counter">
                <s:param name="first" value="1" />
                <s:param name="last" value="#session.contextcount" />
                <!--遍历选项-->
                <s:iterator>
                    <!--投票选项编号-->
                    <s:property value="%{getText('di')}" />
                    <s:property />
                    <s:property value="%{getText('xiang')}" />
                    <!--投票选项内容输入框-->
                    <s:textfield name="context"
                        value="%{getText('pleaseinputthecontextofnewvo-
                        te')}" />
                </s:iterator>
                <!--下一步按钮 -->
                <s:submit key="next" />
            </s:bean>
        </s:form>
    </body>
...
```

25.8　关于管理和查找投票表示层

为了让读者可以快速地理解和掌握投票管理系统，在具体讲解时按照面向应用的方式对该系统分成 4 层：领域模型层、业务层、持久层和表示层。本节将详细介绍如何设计表示层。

该节主要介绍如何设计管理投票操作的表示层，关于管理投票操作的所有请求都由 Struts 2.x 框架的 Action 类来处理。由于要利用 Spring 2.0 框架来解决该层与其他层的耦合问题，所以所有的 Action 类都需要在 applicationContext.xml 文件中实现依赖注入。

25.8.1　管理投票——查看投票信息

当管理员单击系统主页面管理投票链接后，就会利用 Struts 2.0 框架跳转到显示所有投票信息的页面 searchvote.jsp。关于创建投票主题的操作由 Struts 2.0 框架中名为 FindVote 的 Action 类来处理，FindVote.java 的具体内容如代码 25.44 所示。

代码 25.44　查找投票选项内容的 Action：FindVote.java

```
...
public class FindVote extends VoteRoot {
    public String execute() throws Exception {
        List<Vote> list = new ArrayList();
        // 查出所有投票信息并赋给 list
```

```
        list = voteService.findVote();
        list2 = new ArrayList();
        for (int i = 0; i < list.size(); i++) {
            // 取出对应某条投票信息的管理员编号，由管理员编号查出管理员用户名并封装
            到 admin 对象中
            Admin admin = adminService.findNameById(list.get(i).getAdmi-
            nId());
            VoteInfo vInfo = new VoteInfo();
            // 如果投票信息中 publish==1，显示投票状态为开放
            if (list.get(i).getPublish() == 1)
                vInfo.setPublish(getText("open"));
            else
                vInfo.setPublish(getText("close"));
            // 如果投票信息中 type==1，显示投票类型为单选
            if (list.get(i).getType() == 1)
                vInfo.setType(getText("singgleselect"));
            else
                vInfo.setType(getText("multiselect"));
            vInfo.setAdminname(admin.getName());
            vInfo.setCreatedate(list.get(i).getCreatedate());
            vInfo.setTitle(list.get(i).getTitle());
            vInfo.setVoteId(list.get(i).getVoteId());
            // 将查找出的投票信息封装到 list2
            list2.add(vInfo);
        }
        setList2(list2);
        return SUCCESS;
    }
}
```

首先在 applicationContext.xml 文件中配置 FindVote 类。

```
<!--对 FindVote 类进行配置 -->
<bean id="findVote" class="com.cjg.action.vote.FindVote">
    <property name="voteService" ref="voteService" />
    <property name="adminService" ref="adminService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 findVote 的 action-->
<action name="findVote" class="findVote">
    <result>/jsp/vote/showresult.jsp</result>
    <result name="searchnull">/jsp/vote/searchvote.jsp</result>
    <interceptor-ref name="defaultStack" />
    <interceptor-ref name="SessionInterceptor" />
</action>
```

在配置文件 struts.xml 中涉及的页面是 searchvote.jsp，该页面用来显示所有的投票信息，其具体内容如代码 25.45 所示。

代码 25.45　显示投票信息页面：searchvote.jsp

```
...
    <body>
        <!--显示导航链接 -->
        <a href="../frame/main.jsp"><s:property
                value="%{getText('mainpage')}" /> </a>
        <s:property value="%{getText('searchvote')}" />
```

```
        <center>
            <!--显示错误信息-->
            <font color="red"> <s:actionerror /> <s:fielderror /> </font>
            <s:form action="searchVoteValidate" theme="simple">
                <!--显示投票标题-->
                <s:text name="inputsearchstuff" />
                <s:textfield name="title" />
                <!--提交按钮-->
                <s:submit key="submit" />
            </s:form>
        </center>
    </body>
...
```

25.8.2　管理投票——添加投票选项

在设计关于添加投票选项表现层时，利用 Struts 2..0 框架来实现页面的跳转。该操作涉及的页面是添加投票选项的页面 newvote.jsp。关于添加投票选项的操作由 Struts 2.0 框架中名为 AddOneVoteContext 的 Action 类来处理，AddOneVoteContext.java 的具体内容如代码 25.46 所示。

代码 25.46　添加投票选项的 Action：AddOneVoteContext.java

```
...
public class DelVoteContext extends VoteContextRoot {
    private static final String DELCONTEXTERROR = "delContextError";
    public String execute() throws Exception {
        Vote vote = new Vote();
        vote.setVoteId(voteId);
        // 通过投票 ID 查找投票选项信息并封装到 list
        List<Votecontext> list = voteContextService
                .findVoteContextByVoteId(vote);
        // 如果对应某投票 ID 的投票选项数小于或等于 2，则不可继续删除
        if (list.size() > 2) {
            Votecontext voteContext = new Votecontext();
            voteContext.setVotecontextId(votecontextId);
            voteContextService.delVoteContext(voteContext);
            return SUCCESS;
        } else {
            addActionError(getText("lessthantwo"));
            return DELCONTEXTERROR;
        }
    }
}
```

首先在 applicationContext.xml 文件中配置 AddOneVoteContext 类。

```
<!--对 addOneVoteContext 类进行配置 -->
bean id="addOneVoteContext"
    class="com.cjg.action.vote.AddOneVoteContext">
    <property name="voteContextService" ref="voteContextService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 addOneVoteContext 的 Action-->
```

```
<action name="addOneVoteContext" class="addOneVoteContext">
    <result type="redirect-action">
        findOneVote?voteId=${voteId}
    </result>
    <interceptor-ref name="defaultStack" />
    <interceptor-ref name="SessionInterceptor" />
</action>
```

25.8.3　管理投票——删除投票选项

在设计关于删除投票选项表现层时，利用 Struts 2.0 框架来实现页面的跳转。该操作涉及的页面是删除投票选项的页面 newvote.jsp。关于删除投票选项的操作，由 Struts 2.0 框架中名为 DelVoteContext 的 Action 类来处理，DelVoteContext.java 的具体内容如代码 25.47 所示。

代码 25.47　删除投票选项的 Action：DelVoteContext.java

```
...
public class DelVoteContext extends VoteContextRoot {
    private static final String DELCONTEXTERROR = "delContextError";
    public String execute() throws Exception {
        Vote vote = new Vote();
        vote.setVoteId(voteId);
        // 通过投票 ID 查找投票选项信息并封装到 list
        List<Votecontext> list = voteContextService
                .findVoteContextByVoteId(vote);
        // 如果对应某投票 ID 的投票选项数小于或等于 2，则不可继续删除
        if (list.size() > 2) {
            Votecontext voteContext = new Votecontext();
            voteContext.setVotecontextId(votecontextId);
            voteContextService.delVoteContext(voteContext);
            return SUCCESS;
        } else {
            addActionError(getText("lessthantwo"));
            return DELCONTEXTERROR;
        }
    }
}
```

首先在 applicationContext.xml 文件中配置 DelVoteContext 类。

```
<!--对 DelVoteContext 类进行配置 -->
<bean id="delVoteContext"
    class="com.cjg.action.vote.DelVoteContext">
    <property name="voteContextService" ref="voteContextService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 delVoteContext 的 Action-->
<action name="delVoteContext" class="delVoteContext">
    <result type="redirect-action">
        findOneVote?voteId=${voteId}
    </result>
    <result name="delContextError">
        /jsp/vote/managevote.jsp
    </result>
```

```
    <result name="input">
        /jsp/vote/managevote.jsp
</action>
```

25.8.4　管理投票——更新投票主题和投票选项

在设计关于更新投票主题和投票选项表现层时，利用 Struts 2.0 框架来实现页面的跳转。该操作涉及的页面是删除投票选项的页面 newvote.jsp。关于删除投票选项的操作分两步进行：更新投票主题信息和更新投票选项信息，由 Struts 2.0 框架中名为 UpdateVote 和 UpdateVoteContext 的 Action 类来处理。UpdateVote.java 的具体内容如代码 25.48 所示，UpdateVoteContext.java 的具体内容如代码 25.49 所示。

代码 25.48　更新投票选项的 Action：UpdateVote.java

```
...
public class UpdateVote extends VoteRoot {
    public String execute() throws Exception {
        HttpSession session = ServletActionContext.getRequest().getSess-
        ion();
        // 从 Session 中获得投票信息
        Vote vote = (Vote) session.getAttribute("vote");
        // 如果页面请求中的投票类型及投票状态与数据库中信息相符，则不做操作
        if (vote.getType().equals(type) && vote.getPublish().equals(publ-
        ish)) {
            return SUCCESS;
        } else {
            // 否则，将页面请求中的信息更新到数据库
            vote.setType(type);
            vote.setPublish(publish);
            voteService.updateVote(vote);
            return SUCCESS;
        }
    }
}
```

代码 25.49　更新投票选项内容的 Action：UpdateVoteContext.java

```
...
public class UpdateVoteContext extends VoteContextRoot {
    public String execute() throws Exception {
        HttpSession session = ServletActionContext.getRequest().getSess-
        ion();
        // 从 Session 中获得 list
        List<Votecontext> list = (List) session.getAttribute("list");
        for (int i = 0; i < context.length; i++) {
            Votecontext vc = new Votecontext();
            // 如果页面请求中投票子选项信息与 list 中的信息不符，说明有修改
            // 将页面中信息更新到数据库，并将投票子选项票数信息更新为 0
            if (!context[i].equals(list.get(i))) {
                vc.setContext(context[i]);
                vc.setCount(0);
                vc.setVoteId(list.get(i).getVoteId());
                vc.setVotecontextId(list.get(i).getVotecontextId());
                voteContextService.updateVoteContext(vc);
```

```
            } else
                return SUCCESS;
        }
        return SUCCESS;
    }
}
```

首先在 applicationContext.xml 文件中配置 UpdateVote 和 UpdateVoteContext 类。

```
<!--对 UpdateVoteContext 类进行配置 -->
<bean id="updateVoteContext"
    class="com.cjg.action.vote.UpdateVoteContext">
    <property name="voteContextService" ref="voteContextService" />
</bean>
<!--对 UpdateVote 类进行配置 -->
<bean id="updateVote" class="com.cjg.action.vote.UpdateVote">
    <property name="voteService" ref="voteService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 updateVoteContext 的 Action-->
<action name="updateVoteContext" class="updateVoteContext">
    <result type="redirect-action">
        updateVote?type=${type}&amp;publish=${publish}
    </result>
    <interceptor-ref name="defaultStack" />
    <interceptor-ref name="tokenSession" />
    <interceptor-ref name="SessionInterceptor" />
</action>
<!--配置名为 updateVote 的 Action-->
<action name="updateVote" class="updateVote">
    <result type="redirect">findVote.action</result>
    <interceptor-ref name="defaultStack" />
    <interceptor-ref name="SessionInterceptor" />
</action>
```

在关于查看投票信息、添加投票选项、删除投票选项和更新投票主题和投票选项的模块中，都涉及一个名为 managevote.jsp 的页面，该页面用来实现管理投票，具体内容如代码 25.50 所示。

代码 25.50　管理投票页面：managevote.jsp

```
...
    <body>
        <!--显示导航链接 -->
        <s:property value="%{getText('mainpage')}" />
        <s:property value="%{getText('editvote')}" />
        <s:form action="updateVoteContext" theme="simple">
            <s:property value="#session.vote.title" />
            <a href='<s:url action="addOneVoteContext"><s:param name=
            "voteId"
                value="#session.vote.voteId" /></s:url>'><s:text name=
                "addvc" /> </a>
            <s:iterator value="#session.list">
                <td>
                    <!--显示投票内容-->
                    <s:property value="%{getText('votecontext')}" />
                    <s:textfield name="context" value="%{context}" />
```

```
                </td>
                <td>
                    <a href='<s:url action="delVoteContext"><s:param name=
                    "voteId"
                        value="#session.vote.voteId" /><s:param name="vo-
                        tecontextId"
                        value="votecontextId" /></s:url>'><s:text name=
                        "del" /> </a>
                </td>
            </s:iterator>
            <!--投票类型-->
            <s:property value="%{getText('typeofvote')}" />
            <s:radio list="#{'1':'单选','0':'多选'}" name="type" value=
            "#session.vote.type" />
            <!--投票是否开放-->
            <s:property value="%{getText('publishofvote')}" />
            <s:radio list="#{'1':'是','0':'否'}" name="publish" value=
            "#session.vote.publish" />
            <!--提交按钮-->
            <s:submit key="update" />
        </s:form>
    </body>
...
```

25.8.5　查找投票模块

在设计关于查找投票表现层时，利用 Struts 2.0 框架实现页面的跳转。该操作涉及的页面是实现查找功能的 searchvote.jsp 页面，和设计显示查找投票结果的 showresult.jsp 页面。关于查找投票的操作由 Struts 2.0 框架中的名为 SearchVote 的 Action 类来处理，SearchVote.java 的具体内容如代码 25.51 所示。

代码 25.51　查找投票 Action：SearchVote.java

```
...
public class SearchVote extends VoteRoot {
    private static final String SEARCHNULL = "searchnull";
    public String execute() throws Exception {
        Vote vote = new Vote();
        vote.setTitle(title);
        // 通过标题查找投票信息
        List<Vote> list = voteService.findVoteByTitle(vote);
        // list==null 说明没找到
        if (list == null) {
            addActionError(getText("novalidvote"));
            return SEARCHNULL;
        } else {
            list2 = new ArrayList();
            for (int i = 0; i < list.size(); i++) {
                Admin admin = adminService.findNameById(list.get(i)
                        .getAdminId());
                VoteInfo vInfo = new VoteInfo();
                // 如果投票信息中 publish==1，显示投票状态为开放
                if (list.get(i).getPublish() == 1)
                    vInfo.setPublish(getText("open"));
                else
```

```
                    vInfo.setPublish(getText("close"));
                // 如果投票信息中 type==1，显示投票类型为单选
                if (list.get(i).getType() == 1)
                    vInfo.setType(getText("singgleselect"));
                else
                    vInfo.setType(getText("multiselect"));
                vInfo.setAdminname(admin.getName());
                vInfo.setCreatedate(list.get(i).getCreatedate());
                vInfo.setTitle(list.get(i).getTitle());
                vInfo.setVoteId(list.get(i).getVoteId());
                // 将查找出投票信息封装到 list2
                list2.add(vInfo);
            }
            // 将 list2 发送到页面
            setList2(list2);
            return SUCCESS;
        }
    }
}
```

首先在 applicationContext.xml 文件中配置 SearchVote 类。

```
<!--对 SearchVote 类进行配置 -->
<bean id="searchVote" class="com.cjg.action.vote.SearchVote">
    <property name="voteService" ref="voteService" />
    <property name="adminService" ref="adminService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 searchVote 的 action -->
<action name="searchVote" class="searchVote">
    <result>/jsp/vote/showresult.jsp</result>
    <result name="searchnull">/jsp/vote/searchvote.jsp</result>
    <interceptor-ref name="defaultStack" />
    <interceptor-ref name="SessionInterceptor" />
</action>
```

1. 关于查找投票的页面

页面 searchvote.jsp 用来实现查找投票，该页面的具体内容如代码 25.52 所示。

代码 25.52　关于查找投票页面：searchvote.jsp

```
...
    <body>
        <!--显示导航链接 -->
        <a href="../frame/main.jsp"><s:property
                value="%{getText('mainpage')}" /> </a>
        <s:property value="%{getText('searchvote')}" />
        <center>
            <!--显示错误信息 -->
            <s:actionerror /> <s:fielderror />
            <!--表单 -->
            <s:form action="searchVoteValidate" theme="simple">
                <!--输入文本框-->
                <s:text name="inputsearchstuff" />
```

```
                <s:textfield name="title" />
                <!--提交按钮-->
                <s:submit key="submit" />
            </s:form>
        </center>
    </body>
...
```

2. 关于显示查找投票结果的页面

页面 showresult.jsp 用来显示查找投票的结果，该页面的具体内容如代码 25.53 所示。

代码 25.53　关于显示查找投票结果页面：showresult.jsp

```
...
    <body>
        <!--显示导航链接 -->
        <a href="frame/main.jsp"><s:property value="%{getText('mainpage'
        )}" /> </a>
        <s:property value="%{getText('votemanage')}" />
        <!--遍历结果-->
        <s:iterator value="#request.list2" status="st">
            <s:if test="#st.first==true">
                <!--投票 ID-->
                <td><s:property value="%{getText('voteid')}" /></td>
                <!--投票标题-->
                <td><s:property value="%{getText('votetitle')}" /></td>
                <!--投票创建时间-->
                <td><s:property value="%{getText('votecreatedate')}" />
                </td>
                <!--投票创建者-->
                <td><s:property value="%{getText('votecreater')}" /></td>
                <!--投票状态-->
                <td><s:property value="%{getText('votestate')}" /></td>
                <!--投票类型-->
                <td><s:property value="%{getText('typeofvote')}" /></td>
                <!--投票编辑-->
                <td><s:property value="%{getText('edit')}" /></td>
            </s:if>
            <!--输出投票 ID-->
            <td><s:property value="voteId" /></td>
            <!--输出投票标题-->
            <td><s:property value="title" /></td>
            <!--输出投票的创建时间-->
            <td><s:property value="createdate" /></td>
            <!--输出投票的创建人-->
            <td><s:property value="adminname" /></td>
            <!--输出投票的状态-->
            <td><s:property value="publish" /></td>
            <!--输出投票的类型-->
            <td><s:property value="type" /></td>
            <td><s:property value="createdate" /></td>
            <td><s:property value="adminname" /></td>
            <!--超级链接-->
            <td><a href='<s:url action="findOneVote"><s:param name="flag"
            value="1"/>
```

```
                <s:param name="voteId" value="voteId" /></s:url>'>
                <s:property value="%{getText('edit')}" /></a>
            </td>
        </s:iterator>
    </body>
...
```

25.9　关于实现投票操作表示层

为了让读者可以快速地理解和掌握投票管理系统，在具体讲解时按照面向应用的方式对该系统分成 4 层：领域模型层、业务层、持久层和表示层。本节将详细介绍如何设计表示层。

该节主要介绍如何设计前台投票操作的表示层，关于管理投票操作的所有请求都由 Struts 2.x 框架的 Action 类来处理。由于要利用 Spring 2.0 框架来解决该层与其他层的耦合问题，所以所有的 Action 类都需要在 applicationContext.xml 文件中实现依赖注入。

25.9.1　实现投票操作——显示投票主题和投票选项

在设计实现前台投票操作中的显示投票主题和投票选项的表现层时，利用 Struts 2.0 框架来实现页面的跳转。该模块涉及的页面是关于兴趣投票的 interest.htm 页面；关于语言投票的 language.htm 页面和关于显示投票选项的 showvoting.jsp 页面。

关于显示投票主题和投票选项操作，由 Struts 2.0 框架中名为 ShowVoting 和 ShowVotingContext 的 Action 类来处理。ShowVoting.java 的具体内容如代码 25.54 所示，ShowVotingContext.java 的具体内容如代码 25.55 所示。

代码 25.54　显示投票主题 Action：ShowVoting.java

```
...
public class ShowVoting extends VoteRoot {
    private static final String VOTECLOSE = "voteclose";
    public String execute() throws Exception {
        // 通过投票编号查找投票信息
        Vote vote2 = voteService.findVoteById(voteId);
        // 如果 publish!=1,说明投票状态为未开放
        if (vote2.getPublish() != 1) {
            return VOTECLOSE;
        } else {
            HttpSession session = ServletActionContext.getRequest()
                    .getSession();
            // 如果投票状态为开放，将投票信息放入 Session 的 vote 中
            session.setAttribute("vote2", vote2);
            return SUCCESS;
        }
    }
}
```

代码 25.55　显示投票 Action：ShowVotingContext.java

```
public class ShowVotingContext extends VoteContextRoot {
    public String execute() throws Exception {
        HttpSession session = ServletActionContext.getRequest().getSess-
        ion();
        // 从 Session 中获得投票信息，主要是投票编号
        Vote vote = (Vote) session.getAttribute("vote2");
        // 通过投票编号查找对应投票选项信息
        List<Votecontext> list = voteContextService
                .findVoteContextByVoteId(vote);
        map = new HashMap();
        for (int i = 0; i < list.size(); i++) {
            // 将投票选项编号及内容存入 map
            map.put(list.get(i).getVotecontextId(),
list.get(i).getContext());
        }
        // 将 map 及投票类型传送给对应页面
        setMap(map);
        setType(vote.getType());
        return SUCCESS;
    }
}
```

首先在 applicationContext.xml 文件中配置 ShowVoting 和 ShowVotingContext 类。

```
<!--对 ShowVoting 类进行配置 -->
<bean id="showVoting" class="com.cjg.action.vote.ShowVoting">
    <property name="voteService" ref="voteService" />
</bean>
<!--对 ShowVotingContext 类进行配置 -->
<bean id="showVotingContext"
    class="com.cjg.action.vote.ShowVotingContext">
    <property name="voteContextService" ref="voteContextService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 showVoting 的 action-->
<action name="showVoting" class="showVoting">
    <result name="voteclose">/jsp/vote/voteclose.jsp</result>
    <result type="chain">showVotingContext</result>
</action>
<!--配置名为 showVotingContext 的 action-->
<action name="showVotingContext" class="showVotingContext">
    <result>/jsp/vote/showvoting.jsp</result>
</action>
```

1．设关于兴趣投票的页面

页面 interest.htm 用来作为兴趣投票的页面，该页面的具体内容如代码 25.56 所示。

代码 25.56　关于兴趣投票页面：interest.htm

```
...
    <body>
        <iframe
            <!--链接-->
```

```
    src="http://localhost:8080/votesystem/showVoting.action?voteId=1"
            width="450" frameborder="0" height="350" />
    </body>
...
```

2. 设关于语言投票的页面

页面 language.htm 用来作为语言投票的页面，该页面的具体内容如代码 25.57 所示。

代码 25.57　关于语言投票页面：language.htm

```
...
    <body>
        <iframe
            <!--超级链接-->
            src="http://localhost:8080/votesystem/showVoting.action?vot-
            eId=2"
            width="450" frameborder="0" height="350" />
    </body>
...
```

3. 关于显示投票选项的页面

页面 showvoting.jsp 用来作为显示投票选项页面，该页面的具体内容如代码 25.58 所示。

代码 25.58　显示投票选项页面：showvoting.jsp

```
...
    <body>
        <center>
            <s:property value="#session.vote2.title" />
            <s:form action="checkVoter">
                <s:if test="%{type==1}">       <!--判断是多选还是单选-->
                    <!--单选按钮 -->
                    <s:radio list="#request.map" name="context" />
                </s:if>
                <s:else>
                    <!--多选按钮 -->
                    <s:checkboxlist list="#request.map" name="context" />
                </s:else>
                <!--下一步按钮 -->
                <s:submit value="下一步" />
            </s:form>
        </center>
    </body>
...
```

25.9.2　实现投票操作——实现投票处理

在设计实现前台投票操作中的实现投票处理的表现层时，利用 Struts 2.0 框架来实现页面的跳转。关于实现投票处理操作由 Struts 2.0 框架中名为 Voting 的 Action 类来处理，

Voting.java 的具体内容如代码 25.59 所示。

代码 25.59　实现投票处理 Action：Voting.java

```
...
public class Voting extends VoteContextRoot {
    public String execute() throws Exception {
        Votecontext voteContext;
        String vc[] = context;
        for (int i = 0; i < vc.length; i++) {
            voteContext = new Votecontext();
            // 设置投票子选项编号 1
            voteContext.setVotecontextId(Integer.parseInt(vc[i]));
            // 由投票子选项编号查找对应系选项信息
            Votecontext votecontext2 = voteContextService
                    .findVCCountByVCId(voteContext);
            Integer count = votecontext2.getCount();
            String vccontext = votecontext2.getContext();
            votecontext2.setContext(vccontext);
            // 当前投票子选项票数加 1
            votecontext2.setCount(count + 1);
            voteContextService.updateVoteContext(votecontext2);
        }
        return SUCCESS;
    }
}
```

首先在 applicationContext.xml 文件中配置 Voting 类。

```
<!--对 Voting 类进行配置 -->
<bean id="voting" class="com.cjg.action.vote.Voting">
    <property name="voteContextService" ref="voteContextService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 voting 的 action -->
<action name="voting" class="voting">
    <result type="chain">addVoter</result>
</action>
```

25.9.3　实现投票操作——显示投票结果

在设计实现前台投票操作中显示投票结果的表现层时，利用 Struts 2.0 框架来实现页面的跳转。该模块涉及的页面是显示投票结果的页面 showvotingresult.jsp。关于显示投票结果操作由 Struts 2.0 框架中名为 ShowVotingResult 的 Action 类来处理，ShowVotingResult.java 的具体内容如代码 25.60 所示。

代码 25.60　显示投票结果 Action：ShowVotingResult.java

```
...
public class ShowVotingResult extends VoteContextRoot {
    public String execute() throws Exception {
        HttpSession session = ServletActionContext.getRequest().getSess-
        ion();
```

```
        list2 = new ArrayList();
        // 从 Session 中获得投票编号
        Integer vid = ((Vote) session.getAttribute("vote2")).getVoteId();
        // 从数据库中查询得到对应投票编号的总投票数
        Long totalcount = voteContextService.findTotalCountByVoteId(vid);
        // 总投票数存入 Session 的 totalcount
        session.setAttribute("totalcount", totalcount);
        Vote vote = new Vote();
        vote.setVoteId(vid);
        // 通过投票编号查找对应该编号的子选项内容
        List<Votecontext> list = voteContextService
                .findVoteContextByVoteId(vote);
        for (int i = 0; i < list.size(); i++) {
            // 获得子选项投票数，并计算出占该子选项对应投票的总票数的百分比
            Integer count = list.get(i).getCount();
            BigDecimal bd = new BigDecimal((double) count / totalcount);
            double per = bd.setScale(2, BigDecimal.ROUND HALF UP).doub-
            leValue();
            NumberFormat percentFormat = NumberFormat.getPercentInstan-
            ce();
            String percent = percentFormat.format(per);
            VotingInfo votingInfo = new VotingInfo();
            // 设置子选项内容
            votingInfo.setContext(list.get(i).getContext());
            votingInfo.setCount(count);
            // 设置百分比
            votingInfo.setPercent(percent);
            list2.add(votingInfo);
        }
        setList2(list2);
        return SUCCESS;
    }
}
```

首先在 applicationContext.xml 文件中配置 ShowVotingResult 类。

```
<!--对 ShowVotingResult 类进行配置 -->
<bean id="showVotingResult"
    class="com.cjg.action.vote.ShowVotingResult">
    <property name="voteContextService" ref="voteContextService" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 showVotingResult 的 action -->
<action name="showVotingResult" class="showVotingResult">
    <result>/jsp/vote/showvotingresult.jsp</result>
</action>
```

在配置文件 struts.xml 中涉及的页面是 showvotingresult.jsp，该页面用来显示投票结果，其具体内容如代码 25.61 所示。

代码 25.61　显示投票结果页面：showvotingresult.jsp

```
...
        <body>
            <!--显示投票选项的相应信息的标题 -->
            <s:property value="%{getText('thereare')}" />
```

```
            <s:property value="#session.totalcount" />
            <s:property value="%{getText('peoplevoting')}" />
            <s:property value="#session.vote.title" />
            <!--遍历投票结果变量-->
            <s:iterator value="#request.list2">
                <s:property value="context" /></td>    <!--显示投票内容-->
                <s:property value="count" />            <!--显示投票数量-->
                <s:property value="%{getText('piao')}" />  <!--显示“票”-->
                <!--显示百分比的图片-->
                <img src="jsp/img/bar.jpg" width="${count}" height="10" />
                <s:property value="percent" />          <!--显示百分比-->
            </s:iterator>
        </body>
...
```

25.10　小　　结

本章主要介绍一个完整的投票管理系统，该系统实现了标准的 Java EE4 层结构体系，基于 Struts 2.x+Spring+Hibernate 框架构建而成。

在具体实现投票管理系统时，不仅实现了投票管理功能，而且还编写了网上投票功能。在具体实现时，从领域模型层、持久层、业务层和表示层 4 层实现各个模块。其中 Struts 2.x 框架实现表示层页面的跳转，Hibernate 框架实现由数据库记录转变成 POJO 对象的持久层，Spring 框架主要实现该系统业务逻辑的服务层。

第 26 章　权限管理系统（Struts 2.x+Spring+JPA）

在一个完整的网络系统中，管理员需要对各个使用者的权责给予其所需的权限来浏览相应的页面。对于管理员来说，维护使用者和权限间的关系是一件非常不容易的事情，特别是在人员变动的时候。因此一个好的权限管理系统是大型网站中必不可少的系统。本章将通过 Struts 2.x+Spring+JPA 框架技术来实现一个完整的权限管理系统，其中持久层使用 JPA 框架，数据库管理系统则为微软公司的 SQL Service 2000。

26.1　权限管理系统简述

为了让读者可以快速地理解和掌握权限管理系统，在具体讲解时先按照面向应用的方式对该系统进行分层，然后再对各个应用进行 MVC 分层。在整个系统采用的框架中，Struts 2.x 框架用来实现页面跳转，Spring 框架用来管理 Struts 2.x 框架中的 Action，JPA 框架则用来操作实现持久层和控制事务。当 3 个框架具体整合时，对于请求的处理过程如图 26.1 所示。

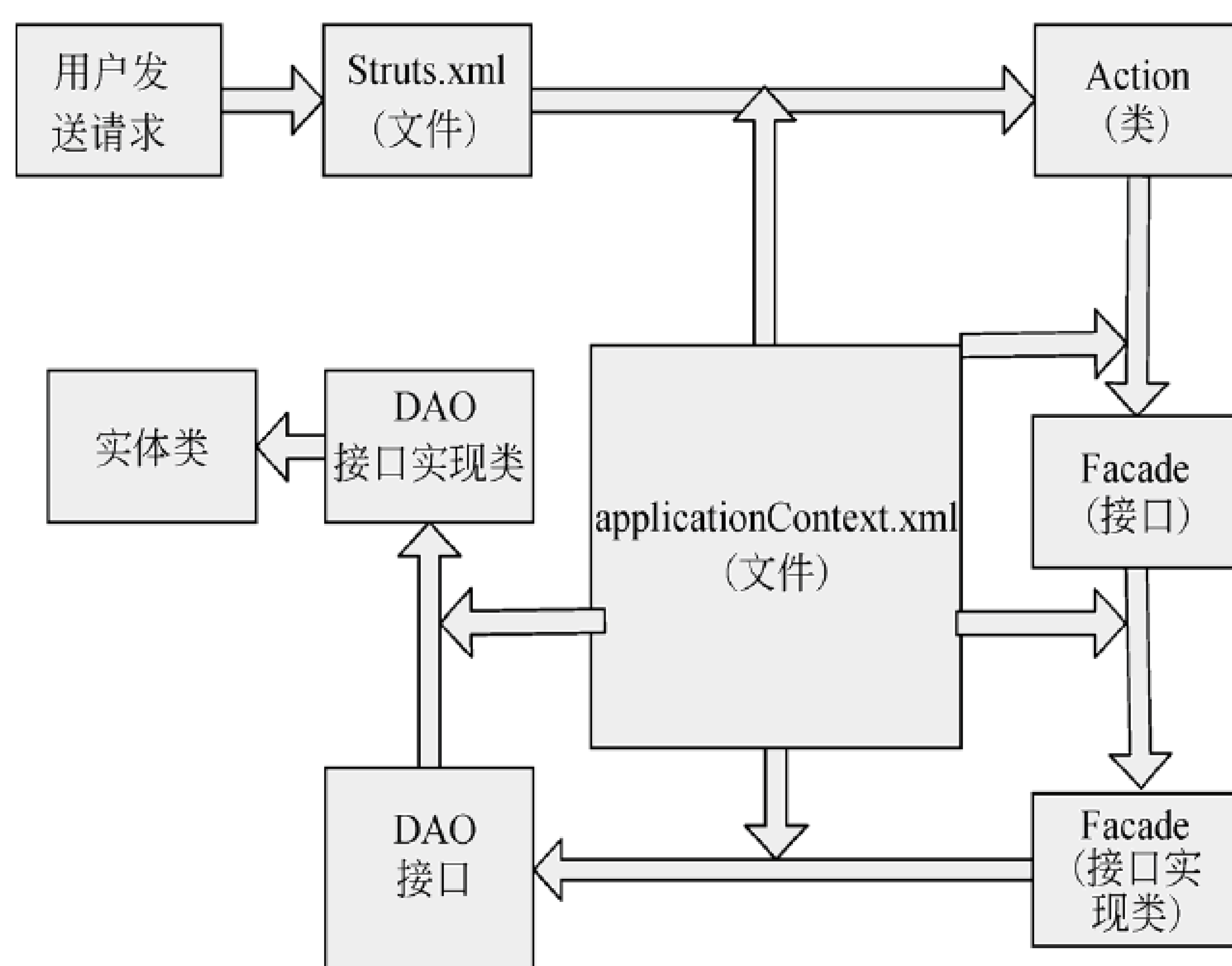


图 26.1　请求的处理过程

26.1.1　权限管理系统的基本原理

1992 年，Ferraiolo 和 Richard.Kuhn 两个人为解决权限管理问题提出了以角色为基础的

存取控制（Role-Based Access-Control，简称 RBAC）理念，该理念主要通过“用户—角色—权限”的关系来控制人员的权限。

RBAC 方式出现以前，开发权限管理系统时主要采用存取控制 ACL 方式来实现。ACL 方式主要通过“使用者——资源”相对应的方式来限制使用者的权限。当资源和程序不是太多的情况下，采用 ACL 方式是一个不错的选择；但是当资源和程序逐渐增加时，一旦人员变动，就需要将与其权限相对应的资源或程序逐一取消和变更。

随着时间的发展，以 RBAC 方式开发的权限管理系统逐渐取代了以 ACL 方式开发的权限管理系统。因为 RBAC 方式可以通过角色来分配用户的权限，即在分配“用户——权限”的关系时，只需要将某一种或多种角色分配给用户，用户就会由这些角色对应的相应权限方式而间接拥有权限。当人员发生变动时，只需要在“用户——角色”或“角色——权限”之间来做适当的变动就可以。

在权限管理系统中，经常会遇到 4 个基本概念，分别为：用户（user）、角色（role）、模块（Module）和功能（Function）。所谓功能就是指一个程序的操作，各个功能之间可以形成树形结构，即每一个功能可以有父功能和子功能。所谓模块就是功能的集合，即多个不能分割的功能定义为一个模块。4 个基本概念的关系如图 26.2 所示。

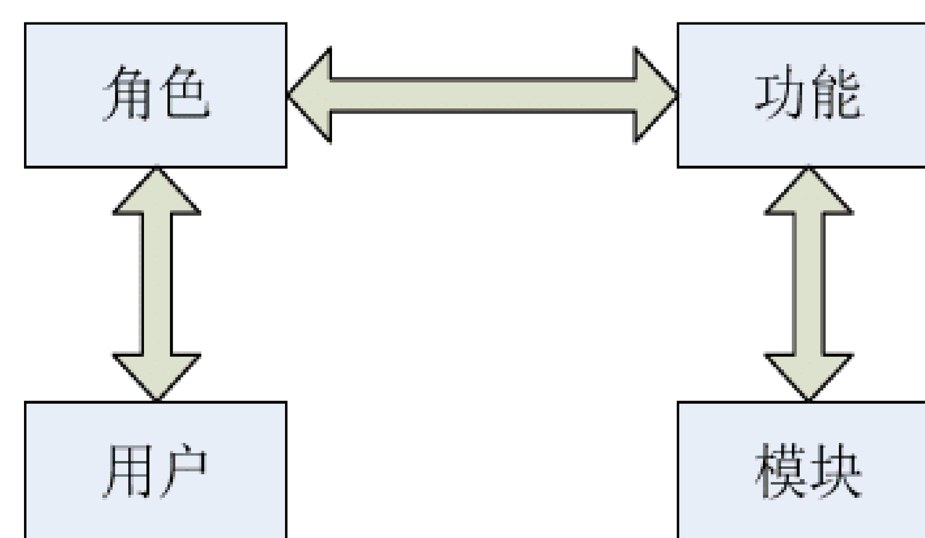


图 26.2　基本概念

26.1.2　权限管理系统描述——管理员

在本节中，以直观的方式来向读者介绍整个权限管理系统要实现的功能。这些功能包括管理员可以访问所有功能、其他用户只能访问规定角色下的功能。

1．管理员的登录

在权限管理系统中，会存在一些初始化信息。例如：超级管理员可以通过用户名和密码（cjgong0，123456）登录该系统（如图 26.3 所示），登录后就会显示关于该系统的初始化模块（如图 26.4 所示）；当单击“维护”模块时，就会出现关于该系统的初始化功能（如图 26.5 所示）；当单击“角色”链接时，就可以出现关于该系统的初始化角色页面（如图 26.6 所示）。



图 26.3　超级管理员登录

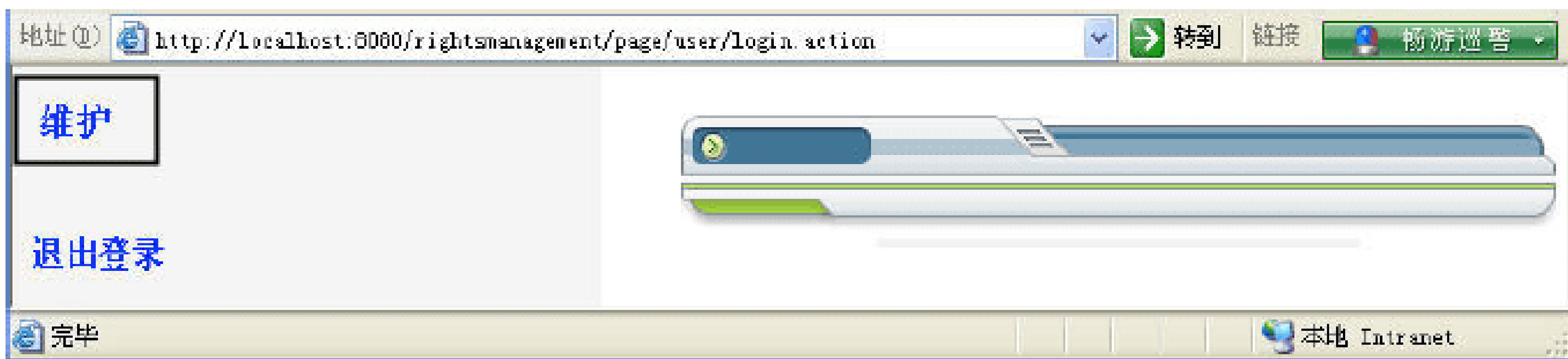


图 26.4　模块页面

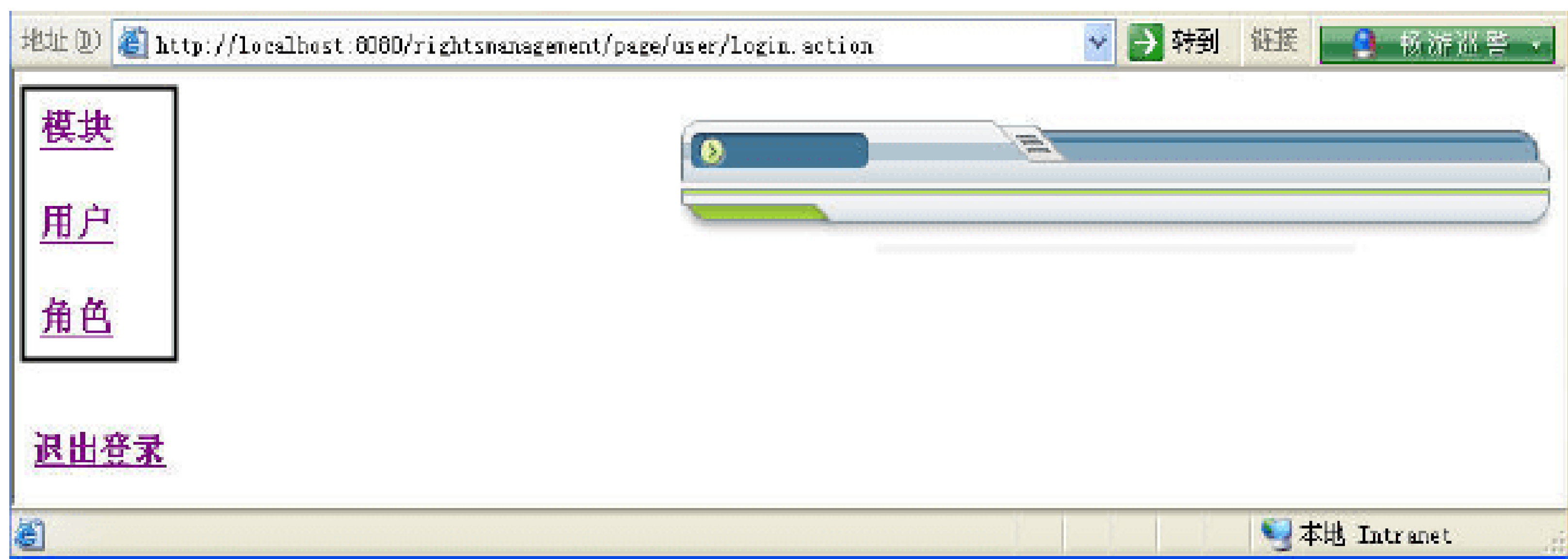


图 26.5　功能页面



图 26.6　角色页面

2．操作功能和模块

在“功能页面”中通过单击“模块”链接，就可以打开如图 26.7 所示关于模块的页面。在该页面中单击“添加”链接就可以打开如图 26.8 所示的“添加模块”对话框。在该对话框中填写完相应信息后，就可以实现添加模块的功能；在该页面中通过单击“修改”链接就可以打开如图 26.9 所示的“修改模块”对话框，在该对话框中可以实现修改模块的功能。



图 26.7　关于模块的页面



图 26.8　添加模块功能

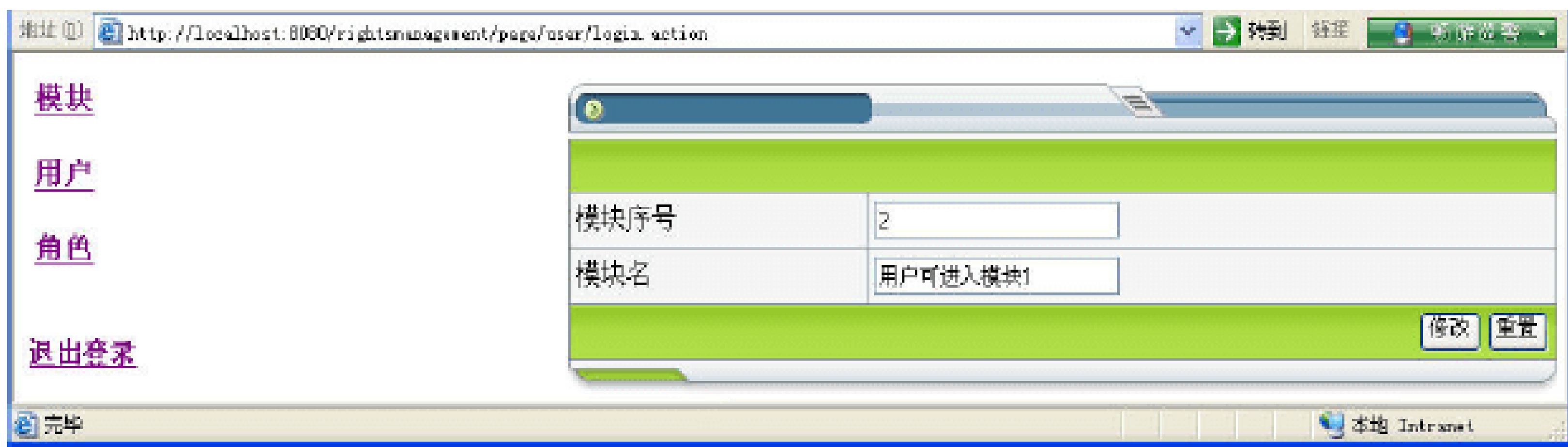


图 26.9　修改模块功能

在关于模块的页面中，通过单击“设置模块功能”链接就可以打开如图 26.10 所示的“设置功能”页面。在该页面中单击“新增”链接就可以打开如图 26.11 所示的“添加模块功能”对话框。在该对话框中填写完相应信息后，就可以实现添加模块功能的功能；在该页面中通过单击关于功能名字的链接就可以打开如图 26.12 所示“修改模块功能”的对话框。在该页面中通过“删除”链接就可以实现删除模块功能的功能。

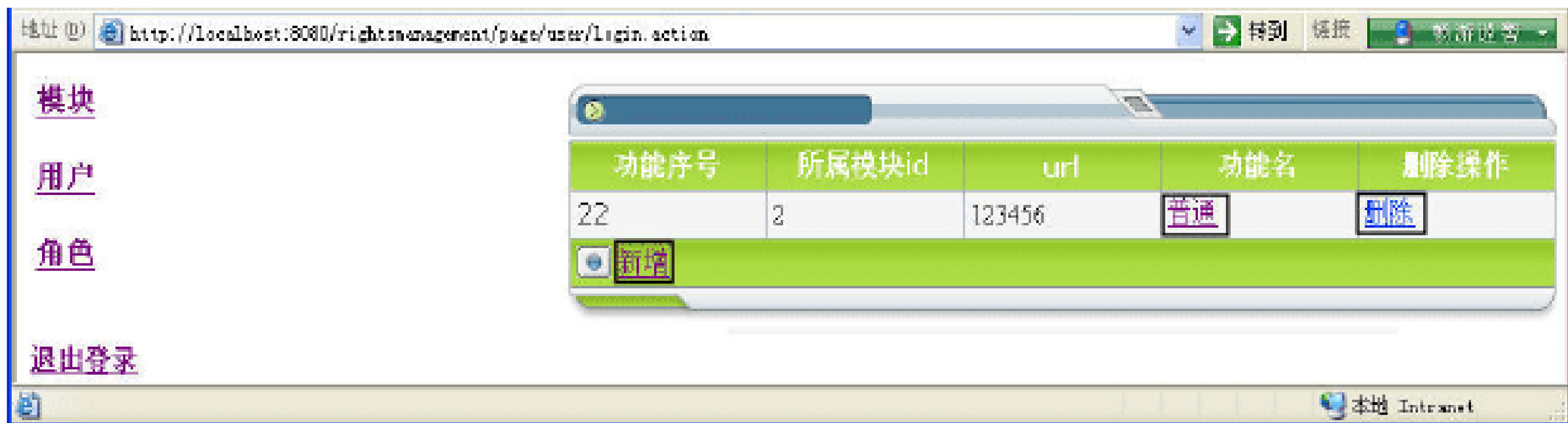


图 26.10　设置功能页面

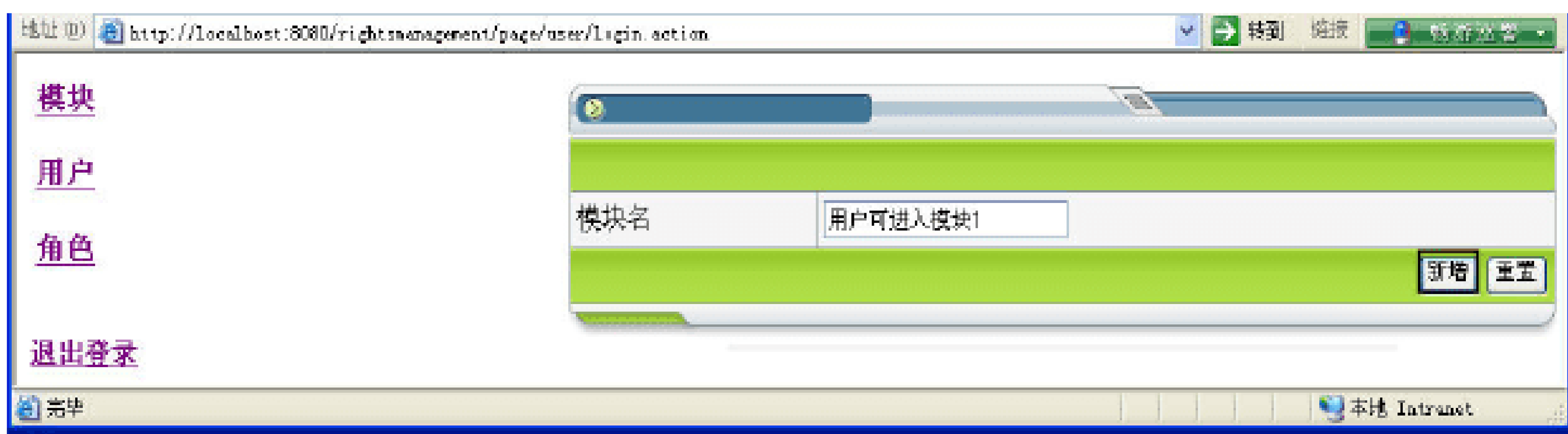


图 26.11　增加模块功能页面

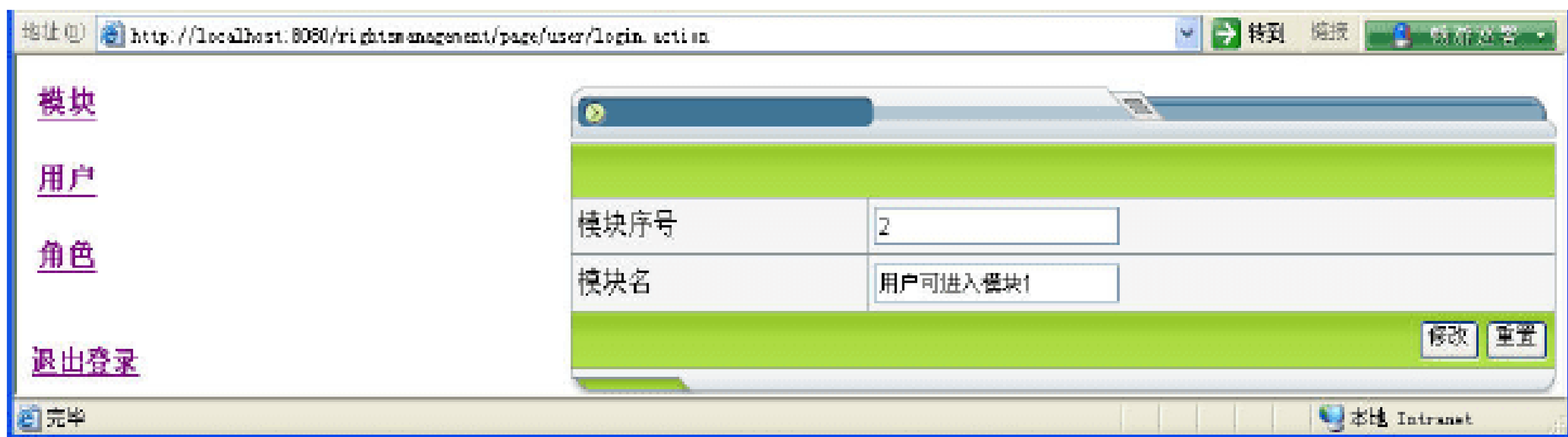


图 26.12　修改模块功能页面

3．操作角色

在“功能页面”中通过单击“角色”链接，就可以打开如图 26.13 所示关于角色的页面。在该页面中单击“添加角色”链接就可以打开如图 26.14 所示的“添加角色”对话框。在该对话框中填写完相应信息后，就可以实现添加角色的功能；在该页面中通过单击“修改”链接就可以打开如图 26.15 所示的“修改角色”对话框，在该对话框中可以实现修改角色的功能；在该页面中通过单击“授权”链接就可以打开如图 26.16 所示的“授权角色”对话框，在该对话框中可以实现为角色授予相应的功能；在该页面中通过“删除”链接就可以实现删除角色的功能。



图 26.13　关于角色的页面



图 26.14　添加角色页面

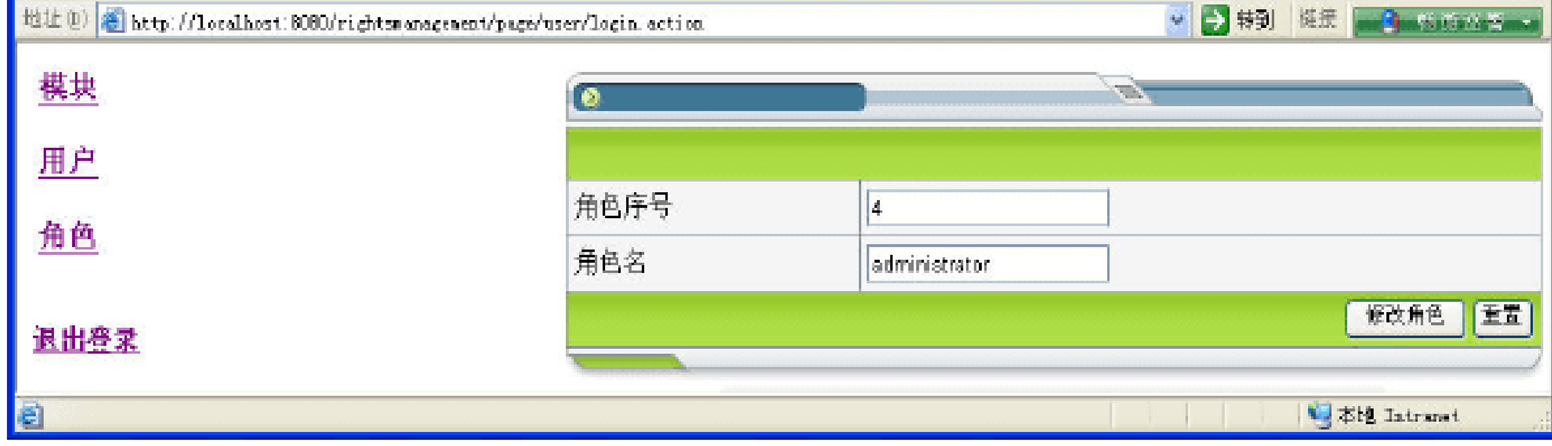


图 26.15　修改角色页面

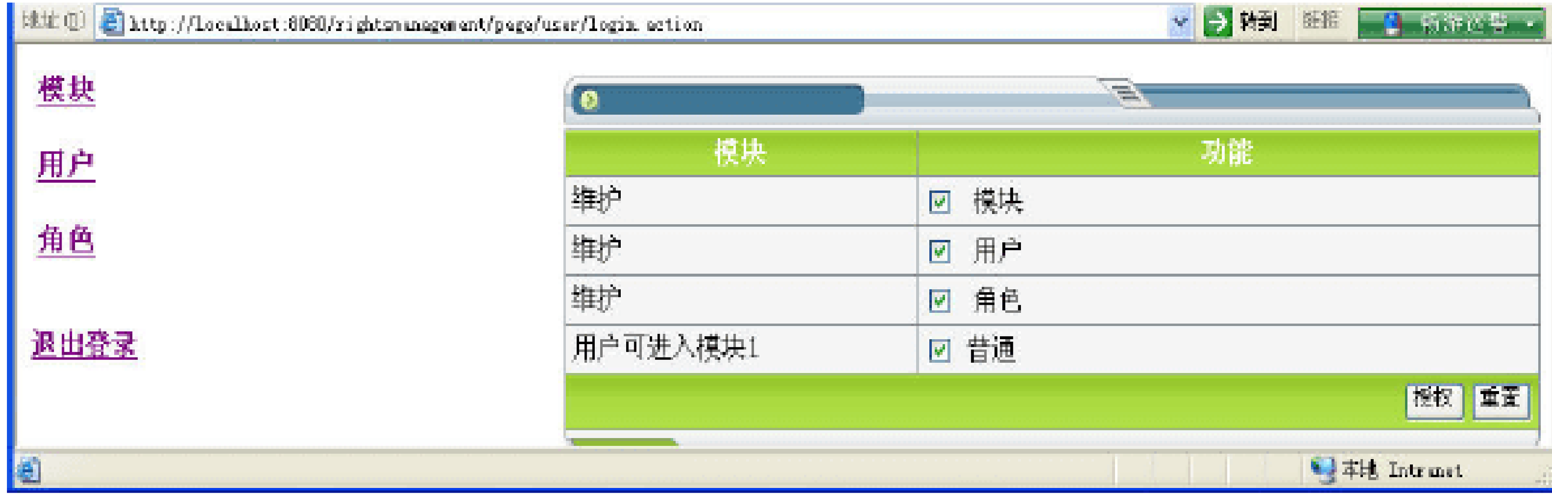


图 26.16　授权角色

4．操作用户

在“功能页面”中通过单击“用户”链接，就可以打开如图 26.17 所示关于用户的页面。在该页面中单击“添加用户”链接就可以打开如图 26.18 所示的“添加用户”对话框。在该对话框中填写完相应信息后，就可以实现添加用户的功能；在该页面中通过单击关于用户名字（cjogng0）的链接就可以打开如图 26.19 所示“修改用户”的页面；在该页面中通过单击“设置角色”链接就可以打开如图 26.20 所示的“设置角色”对话框，在该对话框中通过单击“改变用户角色”按钮就可以实现修改用户角色的功能；在该页面中通过“删除”链接就可以实现删除用户名的功能。



图 26.17　关于用户页面

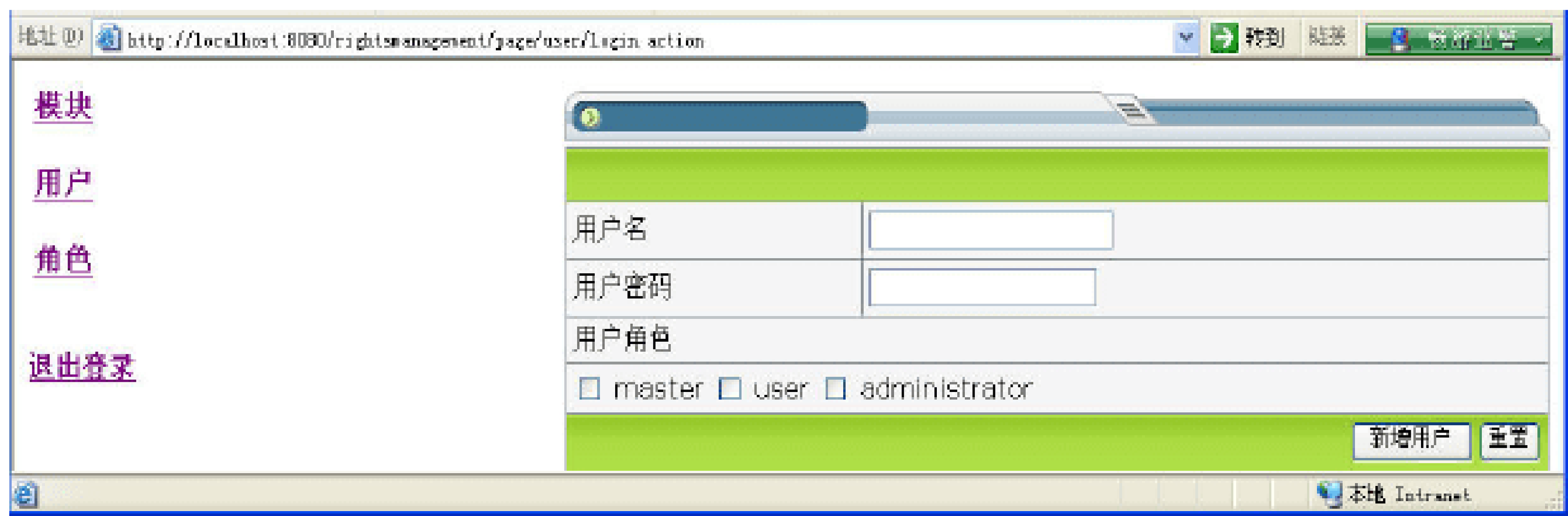


图 26.18　添加用户页面



图 26.19　修改用户页面

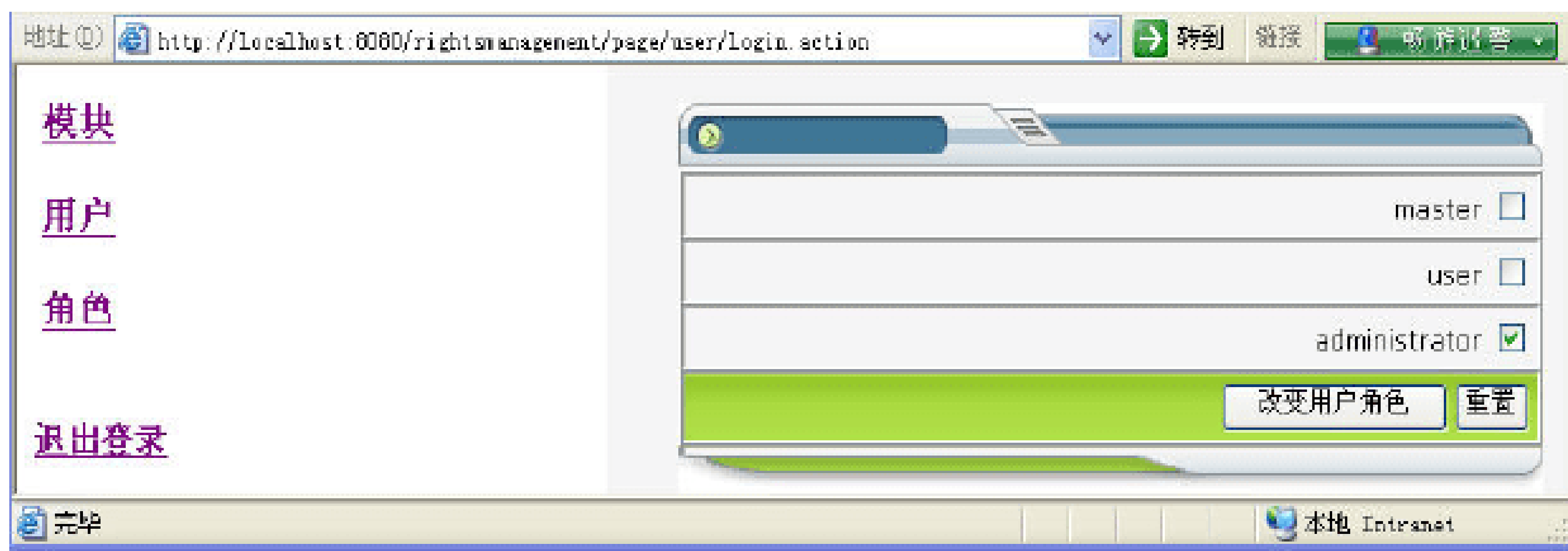


图 26.20　设置角色页面

26.1.3　权限管理系统描述——其他用户

在本节中，以直观的方式来向读者介绍整个权限管理系统要实现的功能。这些功能包括管理员可以访问所有功能、其他用户只能访问规定角色下的功能。

1．cjgong0用户登录

通过用户 cjgong0 登录权限管理系统后，就会进入如图 26.21 所示的关于模块的页面。该用户之所以会拥有上述模块，是因为该用户属于如图 26.22 所示的角色，而该角色却拥有如图 26.23 所示的模块和功能。

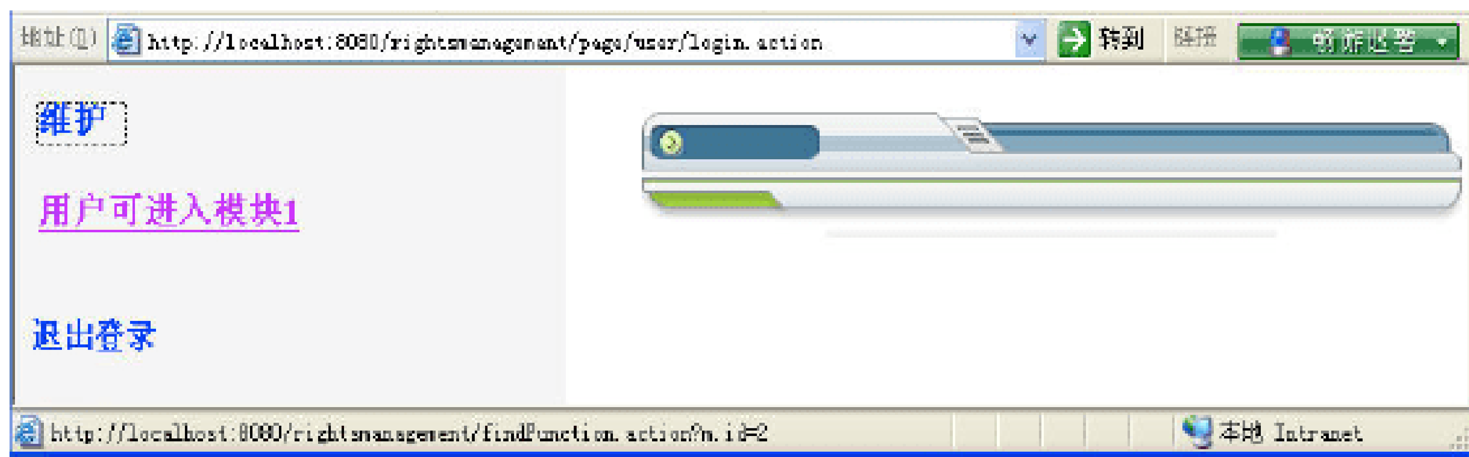


图 26.21　关于 cjgong0 模块

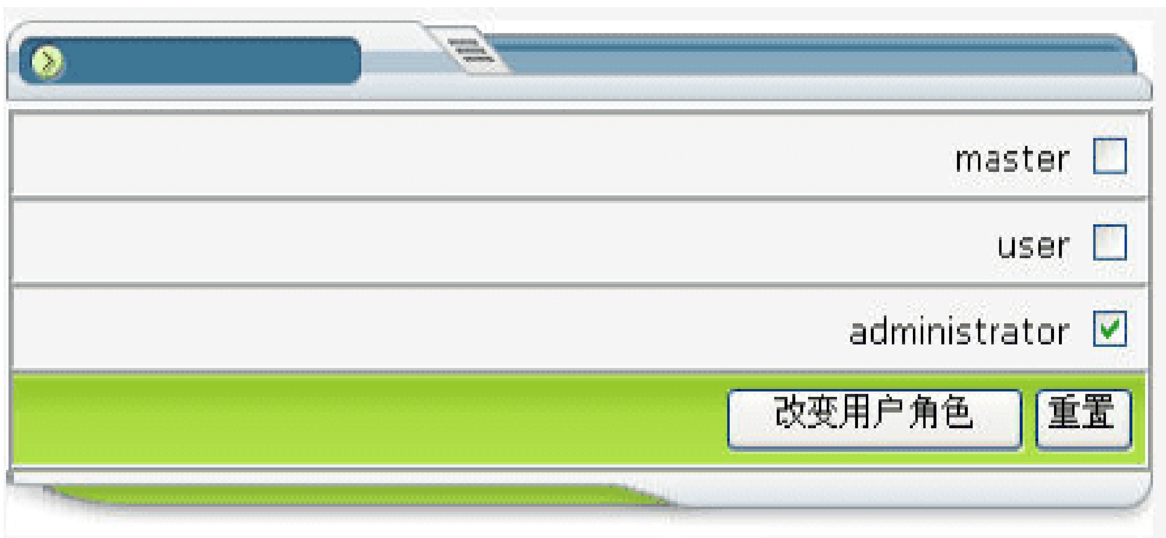


图 26.22　关于 cjgong0 角色

2．cjgong1用户登录

通过用户 cjgong1 登录权限管理系统后，就会进入如图 26.24 所示的关于模块的页面。

该用户之所以会拥有上述模块，是因为该用户属于如图 26.25 所示的角色，而该角色却拥有如图 26.26 所示的模块和功能。



图 26.23　关于 cjgong0 模块和功能

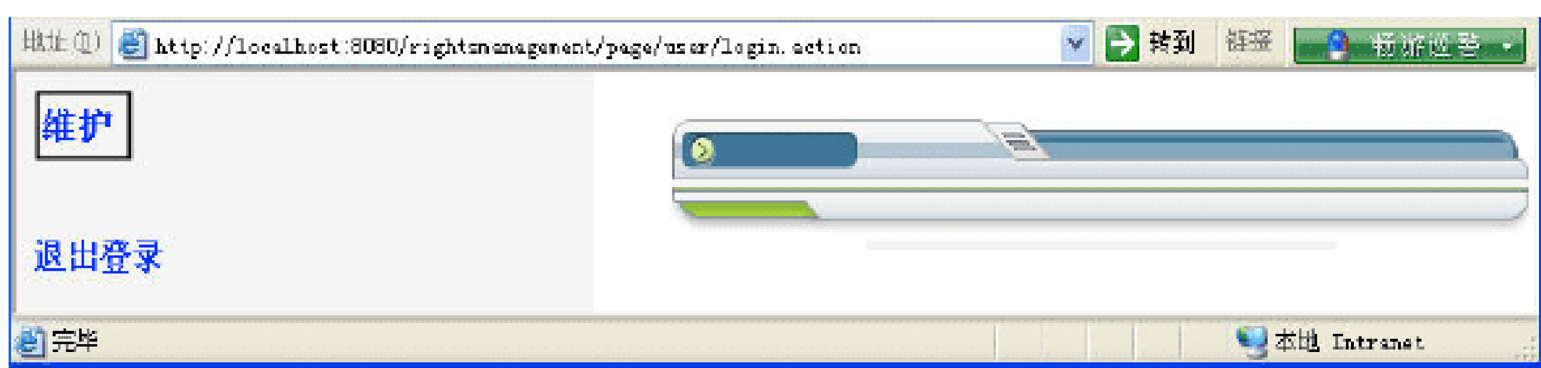


图 26.24　关于 cjgong1 模块



图 26.25　关于 cjgong1 角色



图 26.26　关于 cjgong1 模块和功能

3. cjgong2用户登录

通过用户 cjgong2 登录权限管理系统后，就会进入如图 26.27 所示的关于模块的页面。该用户之所以会拥有上述模块，是因为该用户属于如图 26.28 所示的角色，而该角色却拥有如图 26.29 所示的模块和功能。

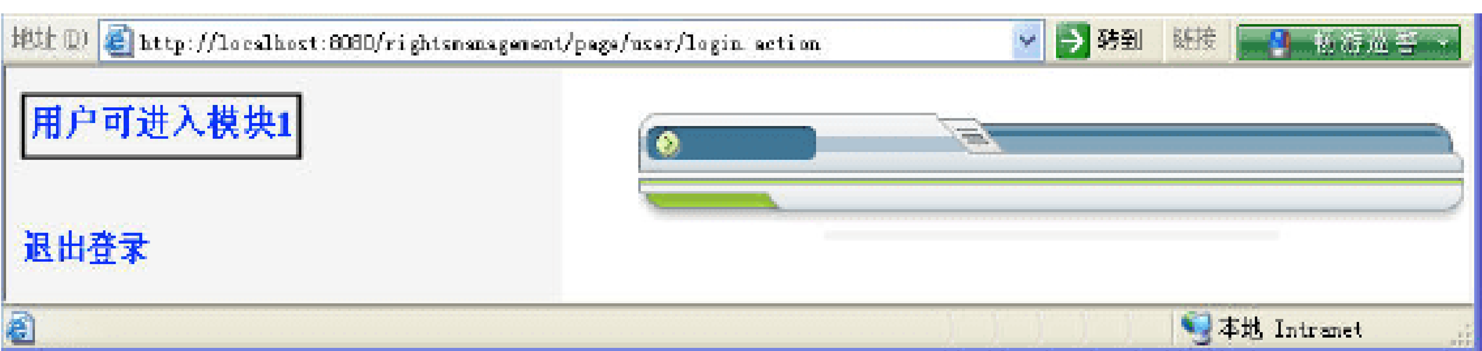


图 26.27　关于 cjgong2 模块

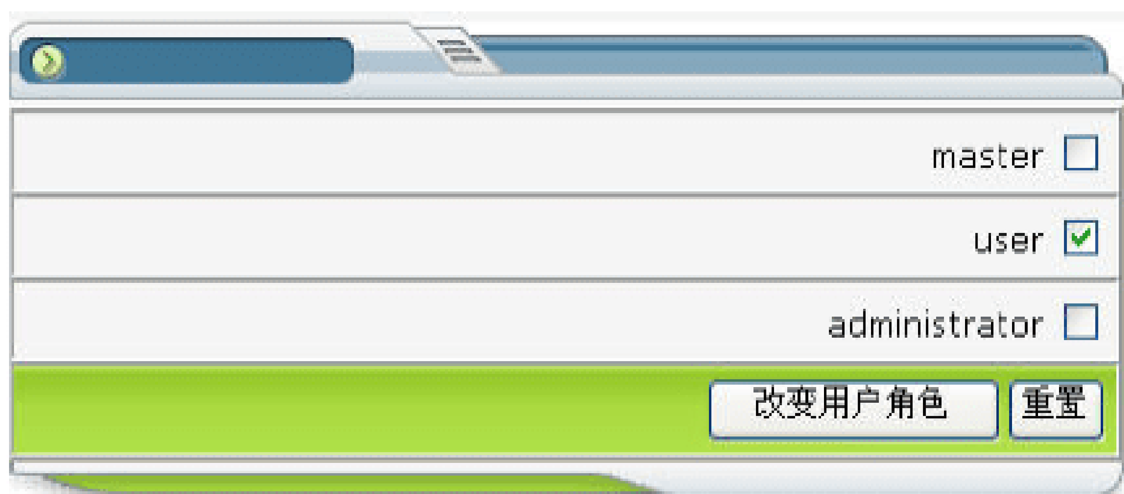


图 26.28　关于 cjgong2 角色



图 26.29　关于 cjgong2 模块和功能

4. cjgong3用户登录

通过用户 cjgong3 登录权限管理系统后，就会进入如图 26.30 所示的关于模块的页面。该用户之所以会拥有上述模块，是因为该用户属于如图 26.31 所示的角色，而该角色却拥有如图 26.32 所示的模块和功能。

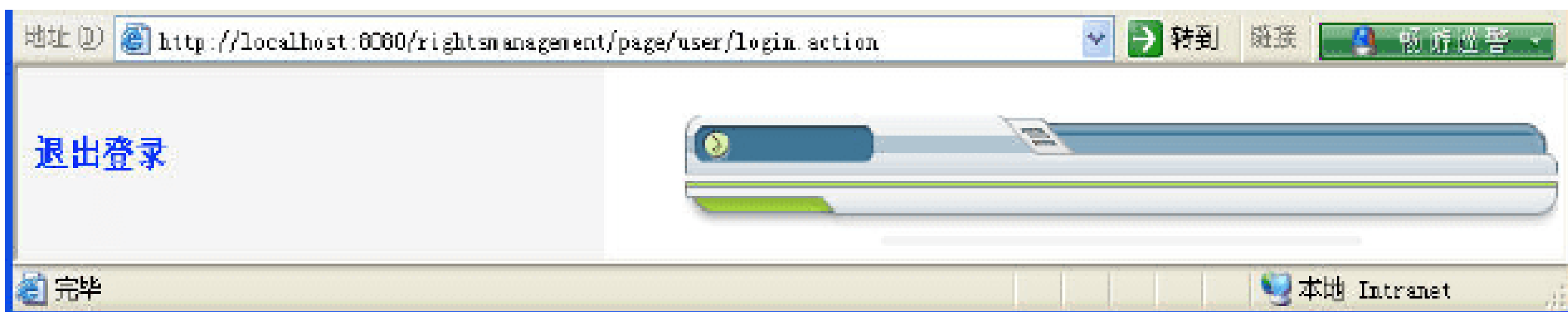


图 26.30　关于 cjgong3 模块

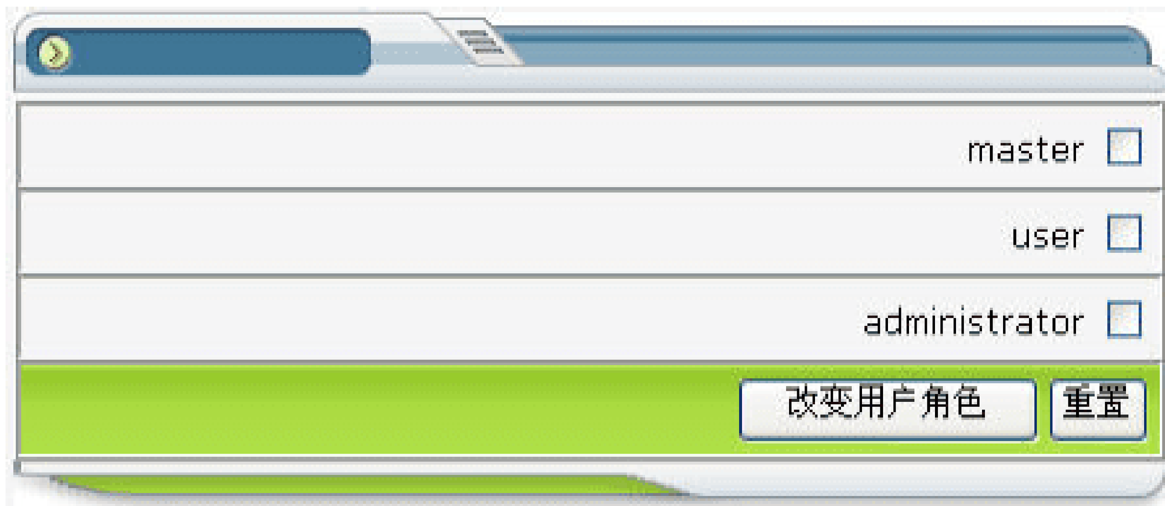


图 26.31　关于 cjgong3 角色

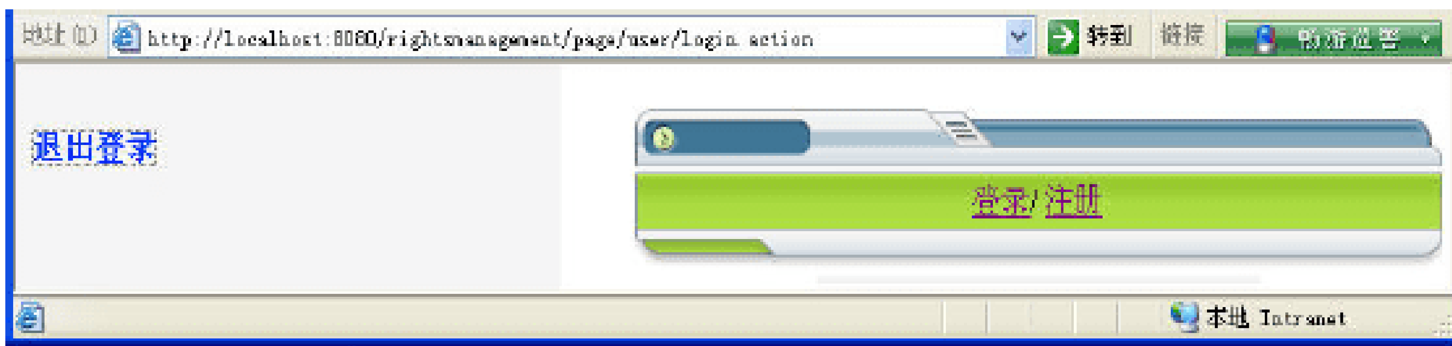


图 26.32　关于 cjgong3 模块和功能

26.2　权限管理系统前期准备

在本节除了将详细地介绍如何设计关于权限管理系统的数据库和表外，还将配置实现该系统将利用的 Struts 2.x+Spring+JPA+SQL Server 2000 框架的环境。其中 Struts 2.x 框架的版本号为 Struts 2.0，Spring 框架的版本号为 Spring 2.5，数据库 Microsoft SQL 为 SQL Server 2000。

26.2.1　设计数据库

权限管理系统需要建立一个数据库并在该数据库中建立 6 张表，分别为存放表的数据库 rightsman、存放用户信息的表 userinfo、存放角色信息的表 role、存放功能信息的表 functions、存放模块信息的表 module、存放角色和功能关系信息的表 role_function 和存放用户和角色关系信息的表 user_role。

1．创建数据库rightsman

如果想创建出数据库 rightsman，可以在 SQL Server 2000 的查询分析器窗口输入如下命令：

```
CREATE DATABASE 'rightsman '
```

2．创建表userinfo

如果想创建出表 userinfo，可以在 SQL Server 2000 的查询分析器窗口输入相关命令。该表的具体信息如表 26.1 所示。

表 26.1　表useinfo信息

字 段 名 称	数 据 类 型	字 段 说 明
id	int	编号
username	varchar(50)	用户名
password	varchar(50)	密码

实现用户模型的内容如代码 26.1 所示。

代码 26.1　用户模型：Userinfo.java

```
...
//把这个类实体化，并设置其对应表
public class Userinfo implements java.io.Serializable {
    // 对应数据表字段的变量
    private Integer id;
    private String username;
    private String password;
    private Set<UserRole> userRoles = new HashSet<UserRole>(0);
                                                    //对应关联变量
    public Userinfo() {                             //空构造方法
    }
    //省略属性 id、username、password 和 userroles 的 set()和 get()方法
...
}
```

接着在 persistence.xml 文件中加入该实体类的配置。

```
<class>com.cjg.user.domain.Userinfo</class>
```

3．创建表role

如果想创建出表 role，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表的具体信息如表 26.2 所示。

表 26.2　表role信息

字 段 名 称	数 据 类 型	字 段 说 明
id	int	编号
rolename	varchar(50)	角色名

实现角色模型的内容如代码 26.2 所示。

代码 26.2　角色模型：Role.java

```
...
public class Role implements java.io.Serializable {
    //创建对应数据表字段的变量
    private Integer id;
    private String rolename;
    //创建 userroles 和 roleFunctions 关联变量
    private Set<UserRole> userRoles = new HashSet<UserRole>(0);
    private Set<RoleFunction> roleFunctions = new HashSet<RoleFunction>
    (0);
    public Role() {                                 //空构造方法
    }
    //省略属性 id、rolename、userroles 和 rolefunctions 的 get()和 set()方法
...
}
```

接着在 persistence.xml 文件中加入该实体类的配置。

```
<class>com.cjg.user.domain.Role</class>
```

4．创建表module

如果想创建出表 module，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表的具体信息如表 26.3 所示。

表 26.3　表module信息

字 段 名 称	数 据 类 型	字 段 说 明
id	int	编号
modulename	varchar(50)	模块名

实现模块模型的内容如代码 26.3 所示。

代码 26.3　模块模型：Module.java

```
...
public class Module implements java.io.Serializable {
    //创建对应数据表字段的变量
    private Integer id;
    private String modulename;
    //创建 functions 对应关联变量
    private Set<Function> functions = new HashSet<Function>(0);
    public Module() {                               //空构造方法
    }
    //省略属性 id、modulename 和 functions 的 get()和 set()方法
...
}
```

接着在 persistence.xml 文件中加入该实体类的配置。

```
<class>com.cjg.user.domain.Module</class>
```

5．创建表functions

如果想创建出表 functions，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表格的具体信息如表 26.4 所示。

表 26.4　表functions信息

字 段 名 称	数 据 类 型	字 段 说 明
id	int	编号
moduleid	int	module 表外键
url	varchar(50)	功能路径
functionname	varchar(50)	功能名

实现功能模型的内容如代码 26.4 所示。

代码 26.4　功能模型：Function.java

```
...
public class Function implements java.io.Serializable {
    //创建对应数据表字段的变量
    private Integer id;
```

```
    private Module module;
    private String url;
    private String functionname;
    //创建 RoleFunction 对应关联变量
    private Set<RoleFunction> roleFunctions = new HashSet<RoleFunction>
    (0);
    public Function() {                                    //创建空构造方法
    }
    //省略属性 module、functionname、id、url、RoleFunction 的 get()和 set()方法
...
}
```

接着在 persistence.xml 文件中加入该实体类的配置。

```
<class>com.cjg.user.domain.Function </class>
```

6. 创建表role_function

如果想创建出表 role_function，可以在 SQL Server 2000 的查询分析器窗口中输入相关命令。该表格的具体信息如表 26.5 所示。

表 26.5　表role_function信息

字 段 名 称	数 据 类 型	字 段 说 明
id	int	编号
roleid	int	userinfo 表外键
functionid	int	Role 表外键

实现角色和功能关系模型的内容如代码 26.5 所示。

代码 26.5　角色和功能关系模型：RoleFunction.java

```
...
public class RoleFunction implements java.io.Serializable {
    //创建对应数据表字段的变量
    private Integer id;
    private Role role;
    private Function function;
    public RoleFunction() {                                //创建空构造方法
    }
    //省略属性 id、role 和 function 的 set 和 get 方法
...
}
```

接着在 persistence.xml 文件中加入该实体类的配置。

```
<class>com.cjg.user.domain. RoleFunction </class>
```

7. 创建表user_role

如果想创建出表 user_role，可以在 SQL Server 2000 的查询分析器窗口输入相关命令。该表格的具体信息如表 26.6 所示。

表 26.6　表user_role信息

字 段 名 称	数 据 类 型	字 段 说 明
id	int	编号
userid	int	userinfo 表外键
roleid	int	Role 表外键

实现用户和角色关系模型的内容如代码 26.6 所示。

代码 26.6　用户和角色关系模型：UserRole.java

```
...
public class UserRole implements java.io.Serializable {
    //创建对应数据表字段的变量
    private Integer id;
    private Userinfo userinfo;
    private Role role;
    public UserRole() {                                    //创建空构造方法
    }
    //省略属性 id、userinfo 和 role 的 get()和 set()方法
...
}
```

接着在 persistence.xml 文件中加入该实体类的配置。

```
<class>com.cjg.user.domain.UserRole</class>
```

在数据库 lams 中 4 张表格的关系如图 26.33 所示。

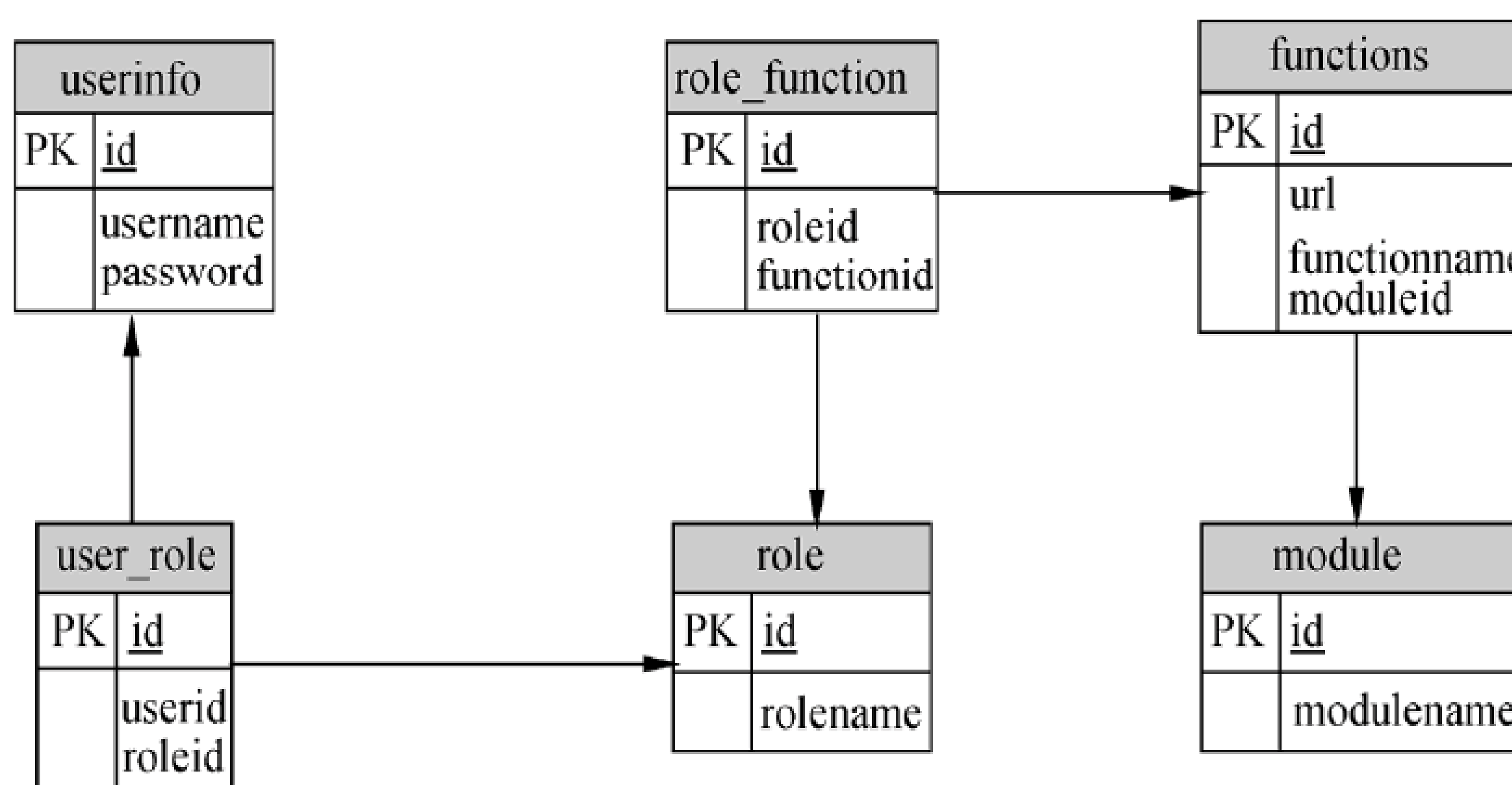


图 26.33　数据库表格的关系

至此，就完成了对权限管理系统数据库和表的设计。

26.2.2　关于 Struts 2.0 的准备

在实现 Struts 2.x 框架、Spring 框架与 JPA 三者集成时，对于其中的 Struts 2.0 框架除了需要引入相应的 jar 包外，还必须要对 struts.xml 和 web.xml 文件做相应的配置。

1．引入jar包

首先引入关于 Struts 2.0 框架的核心包：struts2-core-2.0.11.jar、xwork-2.0.4.jar、

ognl-2.6.11.jar、freemarker-2.3.8.jar 和 commons-logging-1.0.4.jar。然后由于该框架要与 Spring 框架整合，所以还需要 struts2-spring-plugin-2.0.8.jar。最后由于需要连接数据库 MySQL，所以还需要引入关于该数据库的驱动 mysql-connector-java-3.1.14-bin.jar。

注意：前 6 个 jar 包在 Struts 2.0 框架的 jar 包中就可以找到，而最后一个关于数据库的驱动则必须从该数据库的官方网站下载。

2．修改web.xml文件

为了使权限管理系统项目支持 Struts 2.0 框架，需要在 web.xml 文件中增加如代码 26.7 所示的内容。

代码 26.7　修改 web.xml 文件：web.xml

```
...
    <!--实现对 Spring 框架的监听-->
    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    <!--设置过滤器-->
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <!--设置过滤器映射-->
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
...
```

3．创建struts.xml文件

为了使权限管理系统项目支持 Struts 2.0 框架，需要在 votesystem/src 目录下创建 struts.xml 文件，该文件的内容如代码 26.8 所示。

代码 26.8　实现 struts.xml 文件：struts.xml

```
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE struts PUBLIC
       "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
       "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!--定义全局变量 -->
    <constant name="struts.objectFactory" value="spring" />
    <constant name="struts.devMode" value="true" />
    <package name="rightsmanagement" extends="struts-default">
...
    </package>
```

```
</struts>
```

至此，就完成了对权限管理系统中 Struts 2.0 框架的配置。

26.2.3　关于 Spring 2.5 的准备

在实现 Struts 2.x 框架、Spring 框架与 JPA 三者集成时，对于其中的 Spring 2.5 框架除了需要引入相应的 jar 包外，还必须得对 applicationContext.xml 文件做相应的配置。

1．引入jar包

首先引入关于 Spring 2.5 框架的核心包：spring.jar。

2．创建applicationContext.xml文件

首先通过 MyEclipse 开发环境中关于 Spring 框架的向导让权限管理系统支持 Spring 2.5 框架，然后修改 applicationContext.xml 文件支持 JPA 框架。最后 applicationContext.xml 文件的内容如代码 26.9 所示。

代码 26.9　修改 applicationContext.xml 文件：applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http
    ://www.springframework.org/schema/beans/spring-beans-2.0.xsd http:
    //www.springframework.org/schema/tx http://www.springframework.org/
    schema/tx/spring-tx-2.0.xsd"
    xmlns:tx="http://www.springframework.org/schema/tx">
    <!--配置 entityManagerFactory—>
    <bean id="entityManagerFactory"
        class="org.springframework.orm.jpa.LocalEntityManagerFactoryBe-
        an">
        <property name="persistenceUnitName" value="lamsPU" />
    </bean>
    <!--配置 transactionManager—>
    <bean id="transactionManager"
        class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory"
            ref="entityManagerFactory" />
    </bean>
    <tx:annotation-driven transaction-manager="transactionManager" />
</beans>
```

至此，就完成了对权限管理系统中 Spring 2.5 框架的配置。

26.2.4　关于 JPA 的准备

在实现 Struts 2.x 框架、Spring 框架与 JPA 三者集成时，对于其中的 JPA 框架除了需要引入相应的 jar 包外，还必须要创建 persistence.xml 配置文件和 JPA 实体管理器类。

1. 引入jar包

首先通过 MyEclipse 开发环境中关于 JPA 框架的向导，让权限管理系统支持 JPA 框架。

2. 关于persistence.xml文件

当通过 MyEclipse 开发环境中关于 JPA 框架的向导，让权限管理系统支持 JPA 框架后，在 src/META-INF 目录下就会存在一个名为 persistence.xml 的文件。该文件的具体内容如代码 26.10 所示。

代码 26.10　实现 persistence.xml 文件：persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version=
   "1.0">

    <persistence-unit name="lamsPU" transaction-type="RESOURCE_LOCAL">
    <!--配置管理器-->
        <provider>
            oracle.toplink.essentials.PersistenceProvider
        </provider>
        <!--配置类-->
        <class>com.cjg.module.domain.Module</class>
        <class>com.cjg.role.domain.Role</class>
        <class>com.cjg.user.domain.Userinfo</class>
        <class>com.cjg.relationship.domain.RoleFunction</class>
        <class>com.cjg.relationship.domain.UserRole</class>
        <class>com.cjg.functions.domain.Function</class>
        <properties>
            <!--配置数据库驱动-->
            <property name="toplink.jdbc.driver"
                value="com.microsoft.jdbc.sqlserver.SQLServerDriver" />
            <!--配置数据库 URL-->
            <property name="toplink.jdbc.url"
                value="jdbc:microsoft:sqlserver://localhost:1433;Databa-
                seName=rights   man" />
            <!--配置用户名-->
            <property name="toplink.jdbc.user" value="sa" />
            <!--配置密码-->
            <property name="toplink.jdbc.password" value="root" />
        </properties>
    </persistence-unit>
</persistence>
```

3. 创建JPA的实体管理器类EntityManagerHelper

如果想使权限管理系统完全支持 JPA 框架，还需要创建一个实体管理类 EntityManagerHelper.java。该类的具体内容如代码 26.11 所示。

代码 26.11　管理器类：EntityManagerHelper.java

```
...
public class EntityManagerHelper {
```

```
    private static final EntityManagerFactory emf;
                                        //实体化私有静态实体管理器变量 emf
    // 实体化私有静态本地线程变量 threadLocal
    private static final ThreadLocal<EntityManager> threadLocal;
    // 用来给两个变量赋初值的静态块
    static {
        emf = Persistence.createEntityManagerFactory("lamsPU");
        threadLocal = new ThreadLocal<EntityManager>();
    }
    public static EntityManager getEntityManager() {
                                        //得到实体管理器的方法
        EntityManager manager = threadLocal.get();
        if (manager == null || !manager.isOpen()) {
            manager = emf.createEntityManager();
            threadLocal.set(manager);
        }
        return manager;
    }
    public static void closeEntityManager() {      //关闭实体管理器的方法
        EntityManager em = threadLocal.get();
        threadLocal.set(null);
        if (em != null)
            em.close();
    }
    public static void beginTransaction() {        //开始事务的方法
        getEntityManager().getTransaction().begin();
    }
    public static void commit() {                  //提交事务的方法
        getEntityManager().getTransaction().commit();
    }
    public static void rollback() {
        getEntityManager().getTransaction().rollback();    //回滚事务的方法
    }
    public static Query createQuery(String query) {        //生成查找的方法
        return getEntityManager().createQuery(query);
    }
}
```

至此，就完成了对权限管理系统中 JPA 框架的配置。

26.3　权限管理系统具体实现——关联表操作

为了让读者可以快速地理解和掌握权限管理系统，在具体讲解时先按照面向应用的方式对该系统进行分层：关联表操作、模块操作、功能操作、角色操作和用户操作，然后再对各个应用进行 MVC 分层。

本节将详细讲解关联表操作模块，该模块按照 MVC 模式分成为领域模型层和持久层。由于在前面章节已经介绍了领域模型层，所以本节将介绍关于关联表操作模块的持久层。

26.3.1　关于 role_function 关联表操作

在实现关于 role_function 关联表操作时，采用了 DAO 模式。本节将实现对关联表

role_function 操作：增加角色和功能、删除角色和功能，以及查找特定的角色和功能等。

实现操作关联表 role_function 功能的接口如代码 26.12 所示。

代码 26.12　操作 role_function 接口：IRoleFunctionDAO.java

```
...
public interface IRoleFunctionDAO {
    public void save(RoleFunction entity);          //新增数据
    public void delete(RoleFunction entity);        //删除数据
    //通过表中一个字段进行查询
    public List<RoleFunction> findByProperty(String propertyName, Object
    value);
    public List<Module> findModuleHad(Integer i);
                                            //通过角色 ID 查询功能所属模块
    public List<Function> findFInRM(Integer rid, Module pf);
                                            //通过模块 ID 和角色 ID 查询功能
    //通过表中两个字段进行查询
    public List<RoleFunction> findBy2Properties(String propertyName1,
            String propertyName2, final Object value1, final Object value2);
    public List<Integer> findFidByRid(Role r); //通过角色 ID 查询功能 ID
}
```

【代码解析】

- save()方法用来增加角色和功能映射。
- delete()方法用来删除角色和功能映射。
- findByProperty()方法用表中一个字段查询角色和功能映射。
- findBy2Property()方法用表中两个字段查询角色和功能映射。
- findModuleHad()方法用角色的 ID 来查询功能所属模块。
- findFInRM()方法用模块 ID 来查询角色 ID。
- findFidByRid()方法用角色 ID 来查询模块 ID。

实现关于关联表 role_function 接口 IRoleFunctionDAO 的类中，如代码 26.13 所示。

代码 26.13　操作 role_function 接口：RoleFunctionDAO.java

```
...
public class RoleFunctionDAO implements IRoleFunctionDAO {
    private EntityManager getEntityManager() {      //得到实体管理器
        return EntityManagerHelper.getEntityManager();
    }
    public void save(RoleFunction entity) {         //新增数据
        try {
            EntityManagerHelper.beginTransaction();
            getEntityManager().persist(entity);
            EntityManagerHelper.commit();
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public void delete(RoleFunction entity) {       //删除数据
        try {
            EntityManagerHelper.beginTransaction();
            entity = getEntityManager().getReference(RoleFunction.class,
                    entity.getId());
```

```
            getEntityManager().remove(entity);
            EntityManagerHelper.commit();
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    //通过表中一个字段进行查询
    public List<RoleFunction> findByProperty(String propertyName,
            final Object value) {
        try {
            final String queryString = "select model from RoleFunction model
            re model."
                    + propertyName + "= :propertyValue";
            Query query = getEntityManager().createQuery(queryString).
            int(
                    "toplink.refresh", true);
            query.setParameter("propertyValue", value);
            return query.getResultList();
        } catch (RuntimeException re) {
            throw re;
        }
    }
    //通过表中两个字段进行查询
    public List<RoleFunction> findBy2Properties(String propertyName1,
            String propertyName2, final Object value1, final Object value2){
        try {
            final String queryString = "select model from RoleFunction model
            re model."
                    + propertyName1
                    + "= :propertyValue1 and model."
                    + propertyName2 + "=:propertyValue2";
            Query query = getEntityManager().createQuery(queryString).
            Hint(
                    "toplink.refresh", true);
            query.setParameter("propertyValue1", value1).setParameter(
                    "propertyValue2", value2);
            return query.getResultList();
        } catch (RuntimeException re) {
            throw re;
        }
    }
    public List<Module> findModuleHad(Integer rid) {  //通过角色 ID 查询模块
        String queryString = "select distinct rf.function.module from
        RoleFunction rf where rf.role.id=:roleid";
        return getEntityManager().createQuery(queryString).setHint(
                "toplink.refresh", true).setParameter("roleid", rid)
                .getResultList();
    }
    public List<Function> findFInRM(Integer rid, Module m) {
                                                  //通过模块 ID 和角色 ID 查询功能
        String queryString = "select distinct rf.function from RoleFunction
        rf where rf.role.id=:roleid and rf.function.module.id=:mid";
        return getEntityManager().createQuery(queryString).setHint(
                "toplink.refresh", true).setParameter("roleid", rid)
                .setParameter("mid", m.getId()).getResultList();
    }
    public List<Integer> findFidByRid(Role r) {    //通过角色 ID 查询功能 ID
        String  queryString  =  "select  distinct  rf.function.id  from
RoleFunction rf where rf.role.id=:roleid";
```

```
        return getEntityManager().createQuery(queryString).setHint(
                "toplink.refresh", true).setParameter("roleid", r.getId())
                .getResultList();
    }
}
```

在 applicationContext.xml 文件中配置 RoleFunctionDAO 类。

```
<!--对 RoleFunctionDAO 类进行配置 -->
<bean id="roleFunctionDAO" class="com.cjg.relationship.dao.RoleFunction
DAO" />
```

注意：在上述代码中，分别实现了 IRoleFunctionDAO 接口中的所有方法。

26.3.2　关于 user_role 关联表操作

在实现关于 user_role 关联表操作时，采用了 DAO 模式。本节将实现对关联表 user_role 的操作：增加角色和用户、删除角色和用户，以及查找特定的角色和用户等。

实现操作关联表 user_role 功能的接口如代码 26.14 所示。

代码 26.14　操作 user_role 接口：IUserRoleDAO.java

```
...
public interface IUserRoleDAO {
    public void save(UserRole entity);         //新增数据
    public void delete(UserRole entity);       //删除数据
    // 通过表中的一个字段进行查询
    public  List<UserRole>  findByProperty(String  propertyName,  Object
value);
    public List<Integer> findRoleIdByUid(Integer uid);
                                               //通过用户 ID 查找角色 ID
    public List<UserRole> findBy2Properties(String propertyName1,
                                               //通过两个字段进行查找
            final Object value1, String propertyName2, final Object value2);
}
```

【代码解析】

- save()方法用来增加用户和角色映射。
- delete()方法用来删除用户和角色映射。
- findByProperty()方法用表中一个字段查询用户和角色映射。
- findBy2Property()方法用表中两个字段查询用户和角色映射。
- findRoleIdByUid()方法用用户的 ID 来查询用户和角色映射。

实现关于关联表 user_role 接口 IUserRoleDAO 的类如代码 26.15 所示。

代码 26.15　操作 user_role 实现类：UserRoleDAO.java

```
...
public class UserRoleDAO implements IUserRoleDAO {
    private EntityManager getEntityManager() {     //得到实体管理器
        return EntityManagerHelper.getEntityManager();
    }
```

```
public void save(UserRole entity) {           //新增数据
    try {
        EntityManagerHelper.beginTransaction();
        getEntityManager().persist(entity);
        EntityManagerHelper.commit();
    } catch (RuntimeException re) {
        EntityManagerHelper.rollback();
        throw re;
    }
}
public void delete(UserRole entity) {         //删除数据
    try {
        EntityManagerHelper.beginTransaction();
        entity = getEntityManager().getReference(UserRole.class,
                entity.getId());
        getEntityManager().remove(entity);
        EntityManagerHelper.commit();
    } catch (RuntimeException re) {
        EntityManagerHelper.rollback();
        throw re;
    }
}
// 通过表中的一个字段进行查询
public List<UserRole> findByProperty(String propertyName, final Object
value) {
    try {
        final String queryString = "select model from UserRole model
        where model."
                + propertyName + "= :propertyValue";
        Query query = getEntityManager().createQuery(queryString).
        setHint(
                "toplink.refresh", true);
        query.setParameter("propertyValue", value);
        return query.getResultList();
    } catch (RuntimeException re) {
        throw re;
    }
}
// 通过两个字段进行查找
public List<UserRole> findBy2Properties(String propertyName1,
        final Object value1, String propertyName2, final Object value2){
    try {
        final String queryString = "select model from UserRole model
        where model."
                + propertyName1
                + "= :propertyValue1 and model."
                + propertyName2 + "= :propertyValue2";
        Query query = getEntityManager().createQuery(queryString).
        setHint(
                "toplink.refresh", true);
        query.setParameter("propertyValue1", value1).setParameter(
                "propertyValue2", value2);
        return query.getResultList();
    } catch (RuntimeException re) {
        throw re;
    }
}
public List<Integer> findRoleIdByUid(Integer uid) {
                                              //通过用户 ID 查找角色 ID
    String queryString = "select distinct ur.role.id from UserRole ur
```

```
        where ur.userinfo.id=:uid";
        return getEntityManager().createQuery(queryString).setHint(
                "toplink.refresh", true).setParameter("uid", uid)
                .getResultList();
    }
}
```

在 applicationContext.xml 文件中配置 UserRoleDAO 类。

```
<!--对 UserRoleDAO 类进行配置 -->
<bean id="userRoleDAO" class="com.cjg.relationship.dao.UserRoleDAO" />
```

注意：在上述代码中，分别实现了 IUserRoleDAO 接口中的所有方法。

26.4　权限管理系统具体实现——模块操作

为了让读者可以快速地理解和掌握权限管理系统，在具体讲解时先按照面向应用的方式对该系统进行分层：关联表操作、模块操作、功能操作、角色操作和用户操作，然后再对各个应用进行 MVC 分层。

本节将详细讲解模块操作，该模块按照 MVC 模式分成为领域模型层、持久层、业务层和表现层。由于在前面章节已经介绍了领域模型层，所以本节将介绍关于模块操作的其他三层。

26.4.1　模块操作的持久层

在实现关于 module 表操作时，采用了 DAO 模式。本节将实现对表 module 的操作：增加模块、删除模块、修改模块和查询模块。

实现操作表 module 功能的接口如代码 26.16 所示。

代码 26.16　操作表 module 接口：IModuleDAO.java

```
...
public interface IModuleDAO {
    public void save(Module entity);            //新增数据
    public void delete(Module entity);          //删除数据
    public Module update(Module entity);        //修改数据
    public Module findById(Integer id);         //通过 ID 查询数据
    public List<Module> findAll();              //查询所有数据
}
```

【代码解析】

- save()方法用来增加模块信息。
- delete()方法用来删除模块信息。
- update()方法用来更新模块信息。
- findById()方法用来通过 ID 查询模块信息。
- findAll()方法用来查询所有模块信息。

实现关于模块表 Module 接口 IModuleDAO 的类，如代码 26.17 所示。

代码 26.17　操作 Module 实现类：ModuleDAO.java

```
...
public class ModuleDAO implements IModuleDAO {
    public static final String MODULENAME = "modulename";
                                                //创建一个静态常量
    private EntityManager getEntityManager() {     //得到实体管理器
        return EntityManagerHelper.getEntityManager();
    }
    public void save(Module entity) {              //新增数据
        EntityManagerHelper.beginTransaction();
        try {
            getEntityManager().persist(entity);
            EntityManagerHelper.commit();
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public void delete(Module entity) {            //删除数据
        EntityManagerHelper.beginTransaction();
        try {
            entity = getEntityManager().getReference(Module.class,
                    entity.getId());
            getEntityManager().remove(entity);
            EntityManagerHelper.commit();
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public Module update(Module entity) {          //修改数据
        EntityManagerHelper.beginTransaction();
        try {
            Module result = getEntityManager().merge(entity);
            EntityManagerHelper.commit();
            return result;
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public Module findById(Integer id) {           //通过 ID 查询数据
        try {
            Module instance = getEntityManager().find(
                    Module.class, id);
            return instance;
        } catch (RuntimeException re) {
            throw re;
        }
    }
    public List<Module> findAll() {                //查询所有数据
        try {
            final String queryString = "select m from Module m";
            Query query = getEntityManager().createQuery(queryString).
            setHint(
                    "toplink.refresh", true);
```

```
            return query.getResultList();
        } catch (RuntimeException re) {
            throw re;
        }
    }
}
```

在 applicationContext.xml 文件中配置 ModuleDAO 类。

```
<!--对 ModuleDAO 类进行配置 -->
<bean id="moduleDAO" class="com.cjg.module.dao.ModuleDAO" />
```

注意：在上述代码中，分别实现了 IModuleDAO 接口中的所有方法。

26.4.2　模块操作的业务层

在设计实现关于 module 表操作的业务层时，采用了 DAO 模式。本节将在业务层实现对表 module 操作：浏览模块、增加模块、删除模块和修改模块。

业务层实现操作表 module 功能的接口，如代码 26.18 所示。

代码 26.18　操作表 module 接口：IModuleDAO.java

```
...
public interface IModuleFacade {
    public List<Module> findModule(List<Integer> rid); //浏览可进入模块
    public void newModule(Module m);                   //新增模块
    public Module findSingleModule(Integer id);        //单查模块
    public void modifyModule(Module m);                //修改模块
    public List<Module> operateModule();               //浏览全部模块
    public void removeModule(Module m);                //删除模块
}
```

【代码解析】

- findModule ()方法用来浏览指定角色的模块。
- operateModule()方法用来浏览全部模块信息。
- newModule()方法用来增加新的模块信息。
- findSingleModule()方法用来实现单查找模块信息。
- modifyModule()方法用来修改模块信息。
- removeModule()方法用来删除模块信息。

业务层实现关于模块表 Module 接口 IModuleDAO 的类如代码 26.19 所示。

代码 26.19　操作 Module 实现类：ModuleFacade.java

```
...
public class ModuleFacade implements IModuleFacade {
    private IModuleDAO md;
    private IRoleFunctionDAO rfd;
    //省略属性 md 和 rfd 的 get()和 set()方法
...
    public void removeModule(Module m) {            //删除模块
```

```
        md.delete(m);
    }
    public List<Module> findModule(List<Integer> rid) {
                                                    //浏览可进入模块
        List<Module> lm=new ArrayList<Module>();
        for(Integer i:rid){
            lm.addAll(rfd.findModuleHad(i));
        }
        return lm;
    }
    public Module findSingleModule(Integer id) {    //单查模块
        return md.findById(id);
    }
    public void modifyModule(Module m) {            //修改模块
        md.update(m);
    }
    public void newModule(Module m) {               //新增模块
        md.save(m);
    }
    public List<Module> operateModule() {           //浏览全部模块
        return md.findAll();
    }
}
```

在 applicationContext.xml 文件中配置 ModuleFacade 类。

```
<!--对 ModuleFacade 类进行配置 -->
<bean id="moduleFacade" class="com.cjg.module.service.ModuleFacade">
    <property name="md" ref="moduleDAO" />
    <property name="rfd" ref="roleFunctionDAO" />
</bean>
```

注意：在上述代码中，分别实现了 IModuleDAO 接口中的所有方法。

26.4.3　模块操作的表现层

在设计实现关于 module 表操作时的表现层时，利用 Struts 2.0 框架来实现页面的跳转。操作 module 表涉及的页面有：实现查看模块的页面 listModule.jsp、实现查看全部模块的页面 operateModule.jsp、实现新增模块的页面 newModule.jsp 和实现单查模块的页面 listSingleModule.jsp

模块操作的所有请求，都由 Struts 2.0 框架中名为 ModuleAction 的 Action 类来处理，ModuleAction.java 的具体内容如代码 26.20 所示。

代码 26.20　关于模块操作的跳转：ModuleAction.java

```
...
public class ModuleAction {
    //针对于数据库的字段变量
    private Module m;
    private List<Module> lm;
    private IModuleFacade mf;
    public ModuleAction() {                         //构造函数
        m = new Module();
```

```
        lm=new ArrayList<Module>();
    }
    //省略属性 m、lm 和 mf 的 set()和 get()方法
...
    public String findModule() {        //处理 findModule()请求的方法
        HttpSession hs = ServletActionContext.getRequest().getSession();
        lm = mf.findModule((List<Integer>) hs.getAttribute("role"));
        return Action.SUCCESS;
    }
    public String newModule() {         //处理 newModule()请求的方法
        mf.newModule(m);
        return Action.SUCCESS;
    }
    public String findSingleModule() { //处理 findSingleModule()请求的方法
        m = mf.findSingleModule(m.getId());
        return Action.SUCCESS;
    }
    public String modifyModule() {      //处理 modifyModule()请求的方法
        mf.modifyModule(m);
        return Action.SUCCESS;
    }
    public String operateModule() {     //处理 operateModule()请求的方法
        lm = mf.operateModule();
        return Action.SUCCESS;
    }
    public String removeModule() {      //处理 removeModule()请求的方法
        mf.removeModule(m);
        return Action.SUCCESS;
    }
}
```

首先在 applicationContext.xml 文件中配置 ModuleAction 类。

```
<!--对 ModuleAction 类进行配置 -->
<bean id="moduleAction" scope="prototype" class="com.cjg.module.action.
ModuleAction">
    <property name="mf" ref="moduleFacade" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!-- 配置关于 findModule 的 Action-->
<action name="findModule" method="findModule" class="moduleAction">
    <result>/page/module/listModule.jsp</result>
</action>
<!-- 配置关于 operateModule 的 Action-->
<action name="operateModule" method="operateModule" class="moduleAction">
    <result>/page/module/operateModule.jsp</result>
</action>
<!-- 配置关于 modifyModule 的 Action-->
<action name="modifyModule" method="modifyModule" class="moduleAction">
    <result type="chain">operateModule</result>
</action>
<!-- 配置关于 findSingleModule 的 Action-->
<action name="findSingleModule" method="findSingleModule" class="moduleA-
ction">
    <result>page/module/listSingleModule.jsp</result>
</action>
<!-- 配置关于 removeModule 的 Action-->
```

```
<action name="removeModule" method="removeModule" class="moduleAction">
    <result type="chain">operateModule</result>
</action>
<!-- 配置关于 newModule 的 Action-->
<action name="newModule" method="newModule" class="moduleAction">
    <result type="chain">operateModule</result>
</action>
```

1. 设计查看可执行模块页面listModule.jsp

页面 listModule.jsp 用来实现查看模块菜单页，即当用户进入系统中后就会进入该页面，在该页面中显示当前用户指定角色的模块。该页面的具体内容如代码 26.21 所示。

代码 26.21　查看可执行模块页面：listModule.jsp

```
...
    <body>
        <!--输出错误信息 -->
        <s:fielderror></s:fielderror>
        <table>
            <!--遍历模块 -->
            <s:iterator value="lm">
                    <!--输出模块信息 -->
                    <td>
                        <a   href='<s:url   action="findFunction"><s:param
                            name="m.id"                          value="id"
/></s:url>'target="tree">                    <s:property
value="modulename" /> </a>
                    </td>
            </s:iterator>
        </table>
        <!--退出登录链接-->
        <a href="exit.action" target="content"><h4>退出登录</h4></a>
    </body>
...
```

2. 设计查看全部模块的页面operateModule.jsp

页面 operateModule.jsp 用来实现查看全部模块菜单页，在该页面中不仅会显示所有模块，而且还会提供新增页面、单查模块和查看模块功能的请求链接。该页面的具体内容如代码 26.22 所示。

代码 26.22　查看所有模块页面：operateModule.jsp

```
...
        <body>
            <table>
                <!--表格头标题-->
                <TD >模块名</TD >
                <TD >修改</TD >
                <TD >模块功能</TD >
                <!--遍历模块-->
                <s:iterator value="lm">
                    <!--输出模块名字-->
                    <TD ><s:property value="modulename" /></TD>
```

```
                    <!--实现修改功能-->
                    <a href='<s:url action="findSingleModule"> <s:param
                    name="m.id"
                        value="id" /> </s:url>'target="content">修改</a>
                    <!--实现设置模块功能-->
                    <a href='<s:url action="findFByMId"><s:param name=
                    "m.id"
                        value="id" /></s:url>' target="content">设置模块功
                        能</a>
                </s:iterator>
                <!--添加链接-->
                <a href="<%=request.getContextPath()%>/page/module/new-
                Module.jsp"
                    target="content">添加</a>
            </table>
        </body>
...
```

3．设计新增模块的页面newModule.jsp

页面 newModule.jsp 用来实现新增模块页，当在该页面填写完相应的内容后，执行成功后就会转入到查看全部模块页面。该页面的具体内容如代码 26.23 所示。

代码 26.23　新增模块页面：newModule.jsp

```
...
    <body>
        <form action="newModule.action" method="post">
            <table>
                <!--模块名输入框-->
                <TD >模块名
                <s:textfield name="m.modulename" theme="simple" />
                </TD >
                <!--新增和重置按钮-->
                <s:submit value="新增" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </form>
    </body>
...
```

4．设计单查模块的页面listSingleModule.jsp

页面 listSingleModule.jsp 用来实现单查模块功能，通过该页面不仅可以实现单查模块而且还可以修改该模块信息，当执行成功后就会返回到查看全部模块页面。该页面的具体内容如代码 26.24 所示。

代码 26.24　单查模块页面：listSingleModule.jsp

```
...
    <body>
        <form action="modifyModule.action" method="post">
            <table>
                <!--模块序列输入框-->
```

```
                <TD >模块序号
                <s:textfield name="m.id" value="%{m.id}" theme="simple"
                readonly="true" />
                </TD >
                <!--模块名输入框-->
                <TD >模块名
                <s:textfield name="m.modulename" value="%{m.modulename}"
                    theme="simple" />
                </TD >
                <!--修改和重置按钮-->
                <s:submit value="修改" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </form>
    </body>
...
```

26.5　权限管理系统具体实现——功能操作

为了让读者可以快速地理解和掌握权限管理系统，在具体讲解时先按照面向应用的方式对该系统进行分层：关联表操作、模块操作、功能操作、角色操作和用户操作，然后再对各个应用进行 MVC 分层。

本节将详细讲解功能操作，该模块按照 MVC 模式分成为领域模型层、持久层、业务层和表现层。由于在前面章节已经介绍了领域模型层，所以本节将介绍关于功能操作的其他三层。

26.5.1　功能操作的持久层

在实现关于 functions 表操作时，采用了 DAO 模式。本节将实现对表 functions 的操作：增加功能、删除功能、修改功能和查询功能。

实现操作表 functions 功能的接口如代码 26.25 所示。

代码 26.25　操作表 functions 接口：IFunctionDAO.java

```
...
public interface IFunctionDAO {
    public void save(Function entity);              //新增数据
    public void delete(Function entity);            //删除数据
    public Function update(Function entity);        //修改数据
    public Function findById(Integer id);           //通过 ID 查询数据
    // 通过表中一个字段查询数据
    public  List<Function>  findByProperty(String  propertyName,  Object
    value);
    public List<Function> findAll();                //查询所有数据
}
```

【代码解析】

- save()方法用来增加功能信息。

- ❑ delete()方法用来删除功能信息。
- ❑ update()方法用来更新功能信息。
- ❑ findById()方法用来通过 ID 查询功能信息。
- ❑ findAll()方法用来查询所有功能信息。

实现关于功能表 functions 接口 IFunctionDAO 的类，如代码 26.26 所示。

代码 26.26　操作 functions 实现类：FunctionDAO.java

```
...
public class FunctionDAO implements IFunctionDAO {
    // 创建两个静态常量
    public static final String URL = "url";
    public static final String FUNCTIONNAME = "functionname";
    private EntityManager getEntityManager() {      //得到实体管理器
        return EntityManagerHelper.getEntityManager();
    }
    public void save(Function entity) {             //新增数据
        EntityManagerHelper.beginTransaction();
        try {
            getEntityManager().persist(entity);
            EntityManagerHelper.commit();
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public void delete(Function entity) {           //删除数据
        EntityManagerHelper.beginTransaction();
        try {
            entity = getEntityManager().getReference(Function.class,
                    entity.getId());
            getEntityManager().remove(entity);
            EntityManagerHelper.commit();
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public Function update(Function entity) {       //修改数据
        EntityManagerHelper.beginTransaction();
        try {
            Function result = getEntityManager().merge(entity);
            EntityManagerHelper.commit();
            return result;
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public Function findById(Integer id) {          //通过 ID 查询数据
        try {
            Function instance = getEntityManager().find(Function.class,
                    id);
            return instance;
        } catch (RuntimeException re) {
            throw re;
        }
```

```
    }
    // 通过表中的一个字段查询数据
    public List<Function> findByProperty(String propertyName,
            final Object value) {
        try {
            final String queryString = "select model from Function model
            where model."
                    + propertyName + "= :propertyValue";
            Query query = getEntityManager().createQuery(queryString).
            setHint(
                    "toplink.refresh", true);
            query.setParameter("propertyValue", value);
            return query.getResultList();
        } catch (RuntimeException re) {
            throw re;
        }
    }
    public List<Function> findAll() {                //查询所有数据
        try {
            final String queryString = "select model from Function model";
            Query query = getEntityManager().createQuery(queryString).
            setHint(
                    "toplink.refresh", true);
            return query.getResultList();
        } catch (RuntimeException re) {
            throw re;
        }
    }
}
```

在 applicationContext.xml 文件中配置 FunctionDAO 类。

```
<!--对 functionDAO 类进行配置 -->
<bean id="functionDAO" class="com.cjg.functions.dao.FunctionDAO" />
```

注意：在上述代码中，分别实现了 IFunctionDAO 接口中的所有方法。

26.5.2　功能操作的业务层

在设计实现关于 functions 表操作的业务层时，采用了 DAO 模式。本节将在业务层实现对表 functions 的操作：浏览功能、增加功能、删除功能和修改功能。

业务层实现操作表 functions 功能的接口，如代码 26.27 所示。

代码 26.27　操作表 functions 接口：IFunctionFacade.java

```
...
public interface IFunctionFacade {
    public void newFunction(Function f, Integer mid);  //新增功能
    public List<Function> findFunction(List<Integer> rid, Module m);
                                                       //浏览可执行功能
    public void removeFunction(Function f);            //删除功能
    public Function findSingleFunction(Function f);    //单查功能
    public void modifyFunction(Function f);            //修改功能
    public List<Function> findFByMId(Module m);        //浏览全部功能
}
```

【代码解析】

- findFunction()方法用来浏览可执行功能。
- findFByMId()方法用来浏览全部功能信息。
- newFunction()方法用来增加新功能信息。
- findSingleFunction()方法用来实现单查找功能信息。
- modifyFunction()方法用来修改功能信息。
- removeFunction()方法用来删除功能信息。

业务层实现关于功能表 functions 接口 IFunctionFacade 的类，如代码 26.28 所示。

代码 26.28　操作 functions 实现类：FunctionFacade.java

```
...
public class FunctionFacade implements IFunctionFacade {
    //针对于数据库字段的成员变量
    private IRoleFunctionDAO rfd;
    private IFunctionDAO fd;
    private IModuleDAO md;
    //省略属性 rfd、fd 和 md 的 set()和 get()方法
...
    public List<Function> findFunction(List<Integer> rid, Module m) {
                                                    //浏览可执行功能
        List<Function> listfunction = new ArrayList<Function>();
        for (Integer i : rid) {
            listfunction.addAll(rfd.findFInRM(i, m));
        }
        return listfunction;
    }
    public List<Function> findFByMId(Module m) {        //浏览全部功能
        return fd.findByProperty("module.id", m.getId());
    }
    public Function findSingleFunction(Function f) {   //单查功能
        return fd.findById(f.getId());
    }
    public void modifyFunction(Function f) {            //修改功能
        fd.update(f);
    }
    public void newFunction(Function f, Integer mid) {//新增功能
        Module pf = new Module();
        pf = md.findById(mid);
        //设置当前功能所属模块 ID
        f.setModule(pf);
        fd.save(f);
    }
    public void removeFunction(Function f) {            //删除功能
        //删除功能时将其在关联表中的所有数据删除
        for (RoleFunction roleFunction : rfd.findByProperty("function.id",
                f.getId())) {
            rfd.delete(roleFunction);
        }
        fd.delete(f);
    }
}
```

在 applicationContext.xml 文件中配置 FunctionFacade 类。

```
<!--对 functionFacade 类进行配置 -->
```

```
<bean id="functionFacade" class="com.cjg.functions.service.Function
Facade">
    <property name="fd" ref="functionDAO" />
    <property name="md" ref="moduleDAO" />
    <property name="rfd" ref="roleFunctionDAO" />
</bean>
```

注意：在上述代码中，分别实现了 IFunctionFacade 接口中的所有方法。

26.5.3　功能操作的表现层

在设计实现关于 functions 表操作的表现层时，利用 Struts 2.0 框架来实现页面的跳转。操作 functions 表涉及的页面有：实现查看功能的页面 listFunction.jsp、实现查看全部功能的页面 operateFunction.jsp、实现新增功能的页面 newFunction.jsp，以及实现单查功能的页面 listSingleFunction.jsp

模块操作的所有请求都由 Struts 2.0 框架中名为 FunctionAction 的 Action 类来处理，FunctionAction.java 的具体内容如代码 26.29 所示。

代码 26.29　关于功能操作的跳转：FunctionAction.java

```
...
public class FunctionAction {
    //针对于数据库字段的成员变量
    private Function f;
    private Module m;
    private IFunctionFacade ff;
    private List<Function> lf;
    public FunctionAction() {                    //创建构造函数
        f = new Function();
        m = new Module();
    }
    //省略属性 f、m、ff 和 lf 的 set()和 get()方法
...
    public String newFunction() {              //处理 newFunction()请求的方法
        // 将模块 ID 从 Session 中取出
        HttpSession hs = ServletActionContext.getRequest().getSession();
        ff.newFunction(f, (Integer) hs.getAttribute("mid"));
        return Action.SUCCESS;
    }
    public String findFunction() {             //处理 findFunction()请求的方法
        // 将角色信息从 session 中取出
        HttpSession hs = ServletActionContext.getRequest().getSession();
        lf = ff.findFunction((List<Integer>) hs.getAttribute("role"),m);
        return Action.SUCCESS;
    }
    public String removeFunction() {    //处理 removeFunction()请求的方法
        ff.removeFunction(f);
        return Action.SUCCESS;
    }
    public String findSingleFunction() {
                                       //处理 findSingleFunction()请求的方法
        f = ff.findSingleFunction(f);
        return Action.SUCCESS;
    }
```

```
    public String modifyFunction() {    //处理 modifyFunction()请求的方法
        ff.modifyFunction(f);
        return Action.SUCCESS;
    }
    public String findFByMId() {        //处理 findFByMId()请求的方法
        HttpSession hs = ServletActionContext.getRequest().getSession();
        //判断 m 是否为空
        if (m == null || m.getId() == null) {
            //如果 m 为空，则将 session 中的 mid 值赋给 m 的 ID 值
            m.setId((Integer) hs.getAttribute("mid"));
        } else {
            //将模块 ID 做成 Session
            hs.setAttribute("mid", m.getId());
        }
        lf = ff.findFByMId(m);
        ServletActionContext.getRequest().setAttribute("FNo", lf.size());
        return Action.SUCCESS;
    }
}
```

首先在 applicationContext.xml 文件中配置 FunctionAction 类。

```
<!--对 functionAction 类进行配置 -->
<bean id="functionAction" scope="prototype"
    class="com.cjg.functions.action.FunctionAction">
    <property name="ff" ref="functionFacade" />
</bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!-- 配置关于 newFunction 的 Action-->
<action name="newFunction" method="newFunction" class="functionAction">
    <result type="chain">findFByMId</result>
</action>
<!-- 配置关于 findFunction 的 Action-->
<action name="findFunction" method="findFunction" class="functionAction">
    <result>page/functions/listFunction.jsp</result>
</action>
<!-- 配置关于 removeFunction 的 Action-->
<action name="removeFunction" method="removeFunction" class="functionAc
tion">
    <result type="chain">findFByMId</result>
</action>
<!-- 配置关于 findSingleFunction 的 Action-->
<action name="findSingleFunction" method="findSingleFunction" class="func
tionAction">
    <result>/page/functions/listSingleFunction.jsp</result>
</action>
<!-- 配置关于 modifyFunction 的 Action-->
<action name="modifyFunction" method="modifyFunction" class="functionAc-
tion">
    <result type="chain">findFByMId</result>
</action>
<!-- 配置关于 findFByMId 的 Action-->
<action name="findFByMId" method="findFByMId" class="functionAction">
    <result>/page/functions/operateFunction.jsp</result>
</action>
```

1．设计修改功能页面listFunction.jsp

页面 listFunction.jsp 用来实现修改功能，该页面的具体内容如代码 26.30 所示。

代码 26.30　修改功能页面：listFunction.jsp

```
...
    <body>
        <form action="modifyFunction.action" method="post">
        <table>
            <!--输出功能序号 -->
            <TD >功能序号 </TD>
            <TD>
                <s:textfield name="f.id" value="%{f.id}" theme="simple"
                readonly="true"/>
            </TD>
            <!--输出所属模块 ID -->
            <TD>所属模块 id</TD>
            <TD>
                <s:textfield  name="f.module.id"  value="%{f.module.id}"
                    readonly="true" theme="simple"/>
            </TD>
            <!--输出功能名-->
            <TD>功能名</TD>
            <TD >
                <s:textfield name="f.functionname" value="%{f.functionn-
                ame}"theme="simple"/>
            </TD>
            <!--输出功能路径-->
            <TD>功能路径</TD>
            <TD>
                <s:textfield name="f.url" value="%{f.url}" theme="simple"/>
            </TD>
            <TD>
                <!--修改和重置按钮-->
                <s:submit value="修改" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </TD>
        </table>
        </form>
    </body>
...
```

2．设计查看全部功能的页面operateFunction.jsp

页面 operateFunction.jsp 用来实现查看全部功能，在该页面中不仅会显示所有功能，而且还会提供新增功能和删除所有功能的请求链接。该页面的具体内容如代码 26.31 所示。

代码 26.31　查看所有功能页面：operateFunction.jsp

```
...
    <body>
        <table>
            <!--表格头标题-->
            <TD>功能序号</TD>
```

```
            <TD>所属模块 id</TD>
            <TD> url </TD>
            <TD>功能名</TD>
            <TD>删除操作</TD>
            <!--遍历功能-->
            <s:iterator value="lf">
                <!--输出功能 ID-->
                <TD><s:property value="id" /></TD>
                <!--输出功能所属模块 ID-->
                <TD><s:property value="module.id" /></TD>
                <!--输出功能的 URL-->
                <TD><s:property value="url" /></TD>
                <TD>
                <!--输出功能名-->
                <a href='<s:url action="findSingleFunction"><s:param name=
                "f.id" value="id" /></s:url>' target="content"> <s:prop-
                erty value="functionname" /></a>
                </TD>
                <TD>
                <!--实现删除功能-->
                <a href='<s:url action="removeFunction"><s:param name="f.
                id" value="id" /></s:url>' target="content"> 删除 </a>
                </TD>
            </s:iterator>
            <!--新增超级链接-->
            <a href='<%=request.getContextPath()%>/page/functions/new-
            Function.jsp'
                target="content">新增</a>
            <s:if test="#request.FNo==0">
                <!--删除所属模块超级链接-->
                <a href='<s:url action="removeModule"><s:param name="m.id"
                value="m.id" /></s:url>' target="content"> 删除所属模块</a>
            </s:if>
        </table>
    </body>
...
```

3. 设计新增功能的页面newFunction.jsp

页面 newFunction.jsp 用来实现新增功能，当在该页面填写完相应的内容后，执行成功后就会转入到查看全部功能页面。该页面的具体内容如代码 26.32 所示。

代码 26.32　新增功能页面：newFunction.jsp

```
...
    <body>
        <form action="newFunction.action" method="post">
            <table>
                <!--功能名输入框-->
                <TD >功能名
                <s:textfield name="f.functionname" theme="simple" />
                </TD >
                <!--功能路径输入框-->
                <TD >功能路径
                <s:textfield name="f.url" theme="simple" />
                </TD >
```

```
                <!--新增和重置按钮-->
                <s:submit value="新增" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </form>
    </body>
...
```

4．设计单查功能的页面listSingleFunction.jsp

页面 listSingleFunction.jsp 用来实现单查功能，通过该页面不仅可以实现单查功能而且还可以修改该功能信息，当执行成功后就会返回到查看全部功能页面。该页面的具体内容如代码 26.33 所示。

代码 26.33　单查功能页面：listSingleFunction.jsp

```
...
    <body>
        <form action="modifyFunction.action" method="post">
            <table>
                <!--输出功能序号-->
                <TD >功能序号
                <s:textfield name="f.id" value="%{f.id}" theme="simple"
                readonly="true"/>
                </TD >
                <!--输出所属模块 ID -->
                <TD >所属模块 ID -
                <s:textfield  name="f.module.id"  value="%{f.module.id}"
                    readonly="true" theme="simple"/>
                </TD >
                <!--输出功能名-->
                <TD >功能名-
                <s:textfield name="f.functionname" value="%{f.functionn-
                ame}" theme="simple"/>
                </TD >
                <!--输出功能路径 -->
                <TD >功能路径-
                <s:textfield name="f.url" value="%{f.url}" theme="simple"/>
                </TD >
                <!--修改和重置按钮-->
                <s:submit value="修改" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </form>
    </body>
...
```

26.6　权限管理系统具体实现——角色操作

为了让读者可以快速地理解和掌握权限管理系统，在具体讲解时先按照面向应用的方式对该系统进行分层：关联表操作、模块操作、功能操作、角色操作和用户操作，然后再

对各个应用进行 MVC 分层。

本节将详细讲解角色操作，该模块按照 MVC 模式分成为领域模型层、持久层、业务层和表现层。由于在前面章节已经介绍了领域模型层，所以本节将介绍关于角色操作的其他三层。

26.6.1　角色操作的持久层

在实现关于 role 表的操作时，采用了 DAO 模式。本节将实现对表 role 的操作：增加角色、删除角色、修改角色和查询角色。

实现操作表 role 功能的接口如代码 26.34 所示。

代码 26.34　操作表 role 接口：IRoleDAO.java

```
...
public interface IRoleDAO {
    public void save(Role entity);                    //新增数据
    public void delete(Role entity);                  //删除数据
    public Role update(Role entity);                  //修改数据
    public Role findById(Integer id);                 //通过 ID 查询数据
    public List<Role> findAll();                      //查询所有数据
}
```

【代码解析】

- save()方法用来增加角色信息。
- delete()方法用来删除角色信息。
- update()方法用来更新角色信息。
- findById()方法用来通过 ID 查询角色信息。
- findAll()方法用来查询所有角色信息。

实现关于功能表 role 接口 IRoleDAO 的类如代码 26.35 所示。

代码 26.35　操作 role 实现类：RoleDAO.java

```
...
public class RoleDAO implements IRoleDAO {
    public static final String ROLENAME = "rolename";  //声明一个静态常量
    private EntityManager getEntityManager() {         //编写得到实体管理器
        return EntityManagerHelper.getEntityManager();
    }
    public void save(Role entity) {                    //编写新增数据
        try {
            EntityManagerHelper.beginTransaction();
            getEntityManager().persist(entity);
            EntityManagerHelper.commit();
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public void delete(Role entity) {                  //编写删除数据
        try {
            EntityManagerHelper.beginTransaction();
```

```
            entity = getEntityManager()
                    .getReference(Role.class, entity.getId());
            getEntityManager().remove(entity);
            EntityManagerHelper.commit();
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public Role update(Role entity) {                  //编写修改数据
        try {
            EntityManagerHelper.beginTransaction();
            Role result = getEntityManager().merge(entity);
            EntityManagerHelper.commit();
            return result;
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public Role findById(Integer id) {             //编写通过 ID 查询数据
        try {
            Role instance = getEntityManager().find(Role.class, id);
            return instance;
        } catch (RuntimeException re) {
            throw re;
        }
    }
    public List<Role> findAll() {                  //编写查询所有数据
        try {
            final String queryString = "select model from Role model";
            Query query = getEntityManager().createQuery(queryString).
            setHint(
                    "toplink.refresh", true);
            return query.getResultList();
        } catch (RuntimeException re) {
            throw re;
        }
    }
}
```

在 applicationContext.xml 文件中配置 RoleDAO 类。

```
<!--对 RoleDAO 类进行配置 -->
<bean id="functionDAO" class=" com.cjg.role.dao.RoleDAO " />
```

注意：在上述代码中，分别实现了 IRoleDAO 接口中的所有方法。

26.6.2　角色操作的业务层

在设计实现关于 role 表操作时的业务层时，采用了 DAO 模式。本节将在业务层实现对表 role 的操作：浏览角色、增加角色、删除角色、修改角色和对角色授权。

业务层实现操作表 role 功能的接口，如代码 26.36 所示。

代码 26.36　操作表 role 接口：IRoleFacade.java

```
...
public interface IRoleFacade {
    public List<Role> operateRole();                 //浏览全部角色
    public void newRole(Role r);                     //新增角色
    public List<Integer> hadRole(Role r);            //查看当前用户所属角色
    public Role findSingleRole(Role r);              //单查角色
    public void modifyRole(Role r);                  //修改角色
    public void removeRole(Role r);                  //删除角色
    public List<Function> authorization(Role r);     //查找所有角色
    public void changeRf(List<Integer> id, Role r);//对角色授权
}
```

【代码解析】

- ❑ operateRole()方法用来浏览可执行功能。
- ❑ newRole()方法用来浏览全部功能信息。
- ❑ hadRole()方法用来增加新功能信息。
- ❑ findSingleRole()方法用来实现单查找功能信息。
- ❑ modifyRole()方法用来修改功能信息。
- ❑ removeRole()方法用来删除功能信息。
- ❑ authorization()方法查找所有角色信息。
- ❑ changeRf()方法对角色授权。

业务层实现关于功能表 role 接口 IFunctionFacade 的类，如代码 26.37 所示。

代码 26.37　操作表 role 实现类：RoleFacade.java

```
...
public class RoleFacade implements IRoleFacade {
    //针对数据库字段生成属性
    private IRoleFunctionDAO rfd;
    private IFunctionDAO fd;
    private IRoleDAO rd;
    private IUserRoleDAO urd;
    //省略属性 rfd、fd、rd 和 urd 的 get()和 set()方法
...
    public List<Integer> hadRole(Role r) {           //查看当前角色可执行功能
        return rfd.findFidByRid(r);
    }
    public List<Function> authorization(Role r) {       //查找所有功能
        return fd.findAll();
    }
    public void changeRf(List<Integer> id, Role r) {    //对角色授权
        // 判断是否设置角色可执行功能为空
        if (String.valueOf(id.get(0)).equals("ognl.
        NoConversionPossible")) {
            for (RoleFunction rf : rfd.findByProperty("role.id", r.getId())){
                rfd.delete(rf);

            }
        } else {
```

```
            // 删除未被选定的角色可执行功能
            for (Integer j : rfd.findFidByRid(r)) {
                if (!id.contains(j)) {
                    for (RoleFunction rf : rfd.findBy2Properties(
                            "function.id", "role.id", j, r.getId())) {
                        rfd.delete(rf);
                    }
                }
            }
            // 增加选定的角色可执行功能
            for (Integer i : id) {
                if (rfd.findFidByRid(r).contains(i)) {
                    continue;
                } else {
                    RoleFunction rf = new RoleFunction();
                    Function sf = new Function();
                    rf.setRole(r);
                    sf.setId(i);
                    rf.setFunction(sf);
                    rfd.save(rf);
                }
            }
        }
    }

    public Role findSingleRole(Role r) {        //单查角色
        return rd.findById(r.getId());
    }
    public void modifyRole(Role r) {            //修改角色
        rd.update(r);
    }
    public void newRole(Role r) {               //新增角色
        rd.save(r);
    }
    public List<Role> operateRole() {           //浏览全部角色
        return rd.findAll();
    }
    public void removeRole(Role r) {            //删除角色
        // 删除角色时将其在关联表中的所有数据删除
        for (RoleFunction roleFunction : rfd.findByProperty("role.id", r
                .getId())) {
            rfd.delete(roleFunction);
        }
        for (UserRole userRole : urd.findByProperty("role.id", r.getId())){
            urd.delete(userRole);
        }
        rd.delete(r);
    }
}
```

在 applicationContext.xml 文件中配置 RoleFacade 类。

```
    <!--对 RoleFacade 类进行配置 -->
    <bean id="roleFacade" class="com.cjg.role.service.RoleFacade">
        <property name="rd" ref="roleDAO" />
        <property name="rfd" ref="roleFunctionDAO" />
        <property name="fd" ref="functionDAO" />
        <property name="urd" ref="userRoleDAO" />
    </bean>
```

注意：在上述代码中，分别实现了 IRoleFacade 接口中的所有方法。

26.6.3　角色操作的表现层

在设计实现关于 role 表操作的表现层时，利用 Struts 2.0 框架来实现页面的跳转。操作 role 表涉及的页面有：实现查看全部角色信息的页面 operateRole.jsp、实现对角色授权的页面 authorization.jsp、实现新增角色的页面 newRole.jsp 和实现单查角色的页面 listSingleRole.jsp。

角色操作的所有请求，都由 Struts 2.0 框架中名为 RoleAction 的 Action 类来处理，RoleAction.java 的具体内容如代码 26.38 所示。

代码 26.38　关于角色操作的跳转：RoleAction.java

```
...
public class RoleAction {
    //针对于页面的成员变量
    private Role r;
    private List<Role> lr;
    private List<Function> lf;
    private List<Integer> fid;
    private IRoleFacade irf;
    public RoleAction() {                       //构造函数
        fid = new ArrayList<Integer>();
        r = new Role();
    }
    //省略属性 R、Lr、Lf、Fid 和 lrf 的 set()和 get()方法
...
    public String operateRole() {               //处理 operateRole()请求的方法
        lr = irf.operateRole();
        return Action.SUCCESS;
    }
    public String newRole() {                   //处理 newRole()请求的方法
        irf.newRole(r);
        return Action.SUCCESS;
    }
    public String findSingleRole() {    //处理 findSingleRole()请求的方法
        r = irf.findSingleRole(r);
        return Action.SUCCESS;
    }
    public String modifyRole() {                //处理 modifyRole()请求的方法
        irf.modifyRole(r);
        return Action.SUCCESS;
    }

    public String removeRole() {                //处理 removeRole()请求的方法
        irf.removeRole(r);
        return Action.SUCCESS;
    }
    public String authorization() {             //处理 authorization()请求的方法
        ServletActionContext.getRequest().setAttribute("had",
        irf.hadRole(r));
        HttpSession hs = ServletActionContext.getRequest().getSession();
```

```
        hs.setAttribute("r", r);
        lf = irf.authorization(r);
        return Action.SUCCESS;
    }
    public String changeRf() {          //处理 changeRf()请求的方法
        HttpSession hs = ServletActionContext.getRequest().getSession();
        irf.changeRf(fid, (Role) hs.getAttribute("r"));
        return Action.SUCCESS;
    }
}
```

首先在 applicationContext.xml 文件中配置 FunctionAction 类。

```
    <!--对 RoleAction 类进行配置 -->
    <bean id="roleAction" scope="prototype" class="com.cjg.role.action.
    RoleAction">
        <property name="irf" ref="roleFacade" />
    </bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
    <!-- 配置关于 operateRole 的 Action-->
    <action name="operateRole" method="operateRole" class=
    "roleAction">
        <result>/page/role/operateRole.jsp</result>
    </action>
    <!-- 配置关于 newRole 的 Action-->
    <action name="newRole" method="newRole" class="roleAction">
        <result type="chain">operateRole</result>
    </action>
    <!-- 配置关于 findSingleRole 的 Action-->
    <action name="findSingleRole" method="findSingleRole" class=
    "roleAction">
        <result>page/role/listSingleRole.jsp</result>
    </action>
    <!-- 配置关于 modifyRole 的 Action-->
    <action name="modifyRole" method="modifyRole" class="roleAction">
        <result type="chain">operateRole</result>
    </action>
    <!-- 配置关于 removeRole 的 Action-->
    <action name="removeRole" method="removeRole" class="roleAction">
        <result type="chain">operateRole</result>
    </action>
    <!-- 配置关于 authorization 的 Action-->
    <action name="authorization" method="authorization" class=
    "roleAction">
        <result>/page/role/authorization.jsp</result>
    </action>
    <!-- 配置关于 changeRf 的 Action-->
    <action name="changeRf" method="changeRf" class="roleAction">
        <result type="chain">operateRole</result>
    </action>
```

1. 设计查看全部角色信息的页面operateRole.jsp

页面 operateRole.jsp 用来实现查看全部角色信息的菜单页，该页面的具体内容如代码 26.39 所示。

代码 26.39　查看全部角色信息页面：operateRole.jsp

```
...
    <body>
        <form action="modifyRole.action" method="post">
            <table>
                <!--表格头标题 -->
                <TD >角色</TD >
                <TD >修改操作</TD >
                <TD >删除操作</TD >
                <TD >授权操作</TD >
                <!--遍历角色信息 -->
                <s:iterator value="lr">
                    <!--输出角色信息 -->
                    <TD ><s:property value="rolename" /></TD>
                    <!--实现修改功能-->
                    <a href='<s:url action="findSingleRole"><s:param
                    name="r.id"
                         value="id" /></s:url>' target="content">修改</a>
                    <!--实现删除功能 -->
                    <a href='<s:url action="removeRole"><s:param
                    name="r.id"
                        value="id" /></s:url>' target="content">删除</a>
                    <!--实现授权功能-->
                    <a href='<s:url action="authorization"><s:param
                    name="r.id"
                        value="id" /></s:url>' target="content">授权</a>
                </s:iterator>
                <ahref="<%=request.getContextPath()%>/page/role/newRole.
                jsp">
                    添加角色</a>
            </table>
        </form>
    </body>
...
```

2. 设计查看单个模块的页面listSingleRole.jsp

页面 listSingleRole.jsp 用来实现查看单个角色菜单页，该页面的具体内容如代码 26.40 所示。

代码 26.40　查看单个角色页面：listSingleRole.jsp

```
...
    <body>
        <form action="modifyRole.action" method="post">
            <table>
                <!--角色序号输出框-->
                <TD >角色序号
                <s:textfield name="r.id" value="%{r.id}" theme="simple"
                readonly="true" />
                </TD>
                <!--角色名输出框-->
                <TD >角色名
                <s:textfield name="r.rolename" value="%{r.rolename}"
                theme="simple" />
```

```
                </TD>
                <!--提交和重置按钮-->
                <s:submit value="修改角色" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </form>
    </body>
...
```

3．设计新增角色的页面newRole.jsp

页面 newRole.jsp 用来实现新增角色页，该页面的具体内容如代码 26.41 所示。

代码 26.41　新增角色页面：newRole.jsp

```
...
    <body>
        <form action="newRole.action" method="post">
            <table>
                <!--角色名输入框-->
                <TD >角色名
                <s:textfield name="r.rolename" theme="simple"/>
                </TD>
                <!--提交和重置按钮->
                <s:submit value="新增角色" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </form>
    </body>
...
```

4．设计对角色进行授权的页面authorization.jsp

页面 authorization.jsp 用来实现对角色进行授权功能，该页面的具体内容如代码 26.42 所示。

代码 26.42　对角色授权页面：authorization.jsp

```
...
    <body>
        <s:form action="changeRf">
            <table>
                <!--表格头标题-->
                <TD>模块<</TD>
                <TD>功能<</TD>
                <!--遍历权利-->
                <s:iterator value="lf">
                    <!--输出模块-->
                    <TD><s:property value="module.modulename" /></TD>
                    <TD>
                        <!--绑定功能-->
                        <s:if test="id in #request.had">
                        <s:checkbox theme="simple" name="fid" fieldValue=
                        "%{id}"
                                               value="true">
                        </s:checkbox>
```

```
                        <s:property value="%{functionname}" />
                        </s:if>
                        <s:else>
                        <s:checkbox theme="simple" name="fid" fieldValue=
                        "%{id}"
                                                value="false">
                        </s:checkbox>
                        <!--输出功能-->
                        <s:property value="%{functionname}" />
                        </s:else>
                    </TD>
                </s:iterator>
                <!--提交和重置按钮-->
                <s:submit value="授权" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </s:form>
    </body>
...
```

26.7　权限管理系统具体实现——用户操作

为了让读者可以快速地理解和掌握权限管理系统，在具体讲解时先按照面向应用的方式对该系统进行分层：关联表操作、模块操作、功能操作、角色操作和用户操作，然后再对各个应用进行 MVC 分层。

本节将详细讲解用户操作，当该模块按照 MVC 模式分成为领域模型层、持久层、业务层和表现层。由于在前面章节已经介绍了领域模型层，所以本节将介绍关于用户操作的其他三层。

26.7.1　用户操作的持久层

在实现关于 userinfo 表的操作时，采用了 DAO 模式。本节将实现对表 userinfo 的操作：增加用户、删除用户、修改用户和查询用户。

实现操作表 userinfo 功能的接口如代码 26.43 所示。

代码 26.43　操作表 userinfo 接口：IUserinfoDAO.java

```
...
public interface IUserinfoDAO {
    public void save(Userinfo entity);                    //新增数据
    public void delete(Userinfo entity);                  //删除数据
    public Userinfo update(Userinfo entity);              //修改数据
    public Userinfo findById(Integer id);                 //通过 ID 查询数据
    //通过表中的一个字段查询数据
    public List<Userinfo> findByProperty(String propertyName, Object
    value);
    //通过表中 username 字段查询数据
    public List<Userinfo> findByUsername(Object username);
    public List<Userinfo> findAll();                      //查询所有数据
}
```

【代码解析】

- save()方法用来增加用户信息。
- delete()方法用来删除用户信息。
- update()方法用来更新用户信息。
- findById()方法用来通过 ID 查询用户信息。
- findByProperty()方法通过字段来查询用户信息。
- findByUsername()方法通过 Username 字段来查询用户信息。
- findAll()方法用来查询所有用户信息。

实现关于功能表 UserinfoDAO 接口 IUserinfoDAO 的类，如代码 26.44 所示。

代码 26.44　操作 IUserinfoDAO 实现类：UserinfoDAO.java

```
...
public class UserinfoDAO implements IUserinfoDAO {
    //声明静态常量
    public static final String USERNAME = "username";
    public static final String PASSWORD = "password";
    private EntityManager getEntityManager() {      //得到实体管理器
        return EntityManagerHelper.getEntityManager();
    }
    public void save(Userinfo entity) {             //新增数据
        try {
            EntityManagerHelper.beginTransaction();
            getEntityManager().persist(entity);
            EntityManagerHelper.commit();
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public void delete(Userinfo entity) {           //删除数据
        try {
            EntityManagerHelper.beginTransaction();
            entity = getEntityManager().getReference(Userinfo.class,
                     entity.getId());
            getEntityManager().remove(entity);
            EntityManagerHelper.commit();
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
        }
    }
    public Userinfo update(Userinfo entity) {       //修改数据
        try {
            EntityManagerHelper.beginTransaction();
            Userinfo result = getEntityManager().merge(entity);
            EntityManagerHelper.commit();
            return result;
        } catch (RuntimeException re) {
            EntityManagerHelper.rollback();
            throw re;
```

```
        }
    }
    public Userinfo findById(Integer id) {      //通过 ID 查询数据
        try {
            Userinfo instance = getEntityManager().find(Userinfo.
            class, id);
            return instance;
        } catch (RuntimeException re) {
            throw re;
        }
    }
    //通过表中的一个字段查询数据
    public List<Userinfo> findByProperty(String propertyName, final
    Object value) {
        try {
            final String queryString = "select model from Userinfo model
            where model."
                    + propertyName + "= :propertyValue";
            Query query = getEntityManager().createQuery(queryString).
            setHint("toplink.refresh", true);
            query.setParameter("propertyValue", value);
            return query.getResultList();
        } catch (RuntimeException re) {
            throw re;
        }
    }
    //通过表中的 username 字段查询数据
    public List<Userinfo> findByUsername(Object username) {
        return findByProperty(USERNAME, username);
    }
    public List<Userinfo> findAll() {              //查询所有数据
        try {
            final String queryString = "select model from Userinfo model";
            Query query = getEntityManager().createQuery(queryString).
            setHint("toplink.refresh", true);
            return query.getResultList();
        } catch (RuntimeException re) {
            throw re;
        }
    }
}
```

在 applicationContext.xml 文件中配置 UserinfoDAO 类。

```
    <bean id="roleDAO" class="com.cjg.role.dao.RoleDAO" />
```

注意：在上述代码中，分别实现了 IUserinfoDAO 接口中的所有方法。

26.7.2　用户操作的业务层

在设计实现关于 userinfo 表操作的业务层时，采用了 DAO 模式。本节将在业务层实现

对表 userinfo 的操作：浏览角色、增加角色、删除角色、修改角色和对角色授权。

业务层实现操作表 userinfo 功能的接口，如代码 26.45 所示。

代码 26.45　操作表 userinfo 接口：IUserinfoFacade.java

```
...
public interface IUserinfoFacade {
    public String login(Userinfo u);            //用户登录判断用户名和密码
    public String regist(Userinfo u);           //用户注册判断用户名
    public List<Userinfo> operateUser();        //查看所有用户
    public Userinfo findSingleUser(Userinfo u);//单查用户
    public void modifyUser(Userinfo u);         //修改用户
    public String newUser(Userinfo u, List<Integer> lrid);
                                                //新增用户
    public void removeUser(Userinfo u);         //删除用户
    public List<Integer> getRole(Userinfo u);  //查看用户所属角色
    public String changeUR(List<Integer> lrid, Userinfo u);
                                                //改变用户所属角色
}
```

【代码解析】

- login()方法用来判断用户名和密码，实现登录功能。
- regist()方法用来判断用户名，实现注册功能。
- operateUser()方法查看所有用户的功能。
- findSingleUser()方法实现单查用户的功能。
- modifyUser()方法实现修改用户的功能。
- newUser()方法实现增加用户的功能。
- removeUser()方法实现删除用户的功能。
- getRole()方法实现查看用户所属角色的功能。
- changeUR()方法实现改变用户所属角色的功能。

业务层实现关于用户表 userinfo 接口 IUserinfoFacade 的类，如代码 26.46 所示。

代码 26.46　操作表 userinfo 实现类：UserinfoFacade.java

```
...
public class UserinfoFacade implements IUserinfoFacade {
    //针对数据库字段生成属性
    private IUserRoleDAO urd;
    private IUserinfoDAO ud;
    private IRoleDAO rd;
    //省略属性 rd、ud 和 urd 的 get()和 set()方法
...
    // 改变用户所属角色
    public String changeUR(List<Integer> lrid, Userinfo u) {
        if (String.valueOf(lrid.get(0)).equals("ognl.N
        oConversionPossible")) {
            return "input";
        }
        UserRole ur = new UserRole();
        Role r = new Role();
        for (Integer j : urd.findRoleIdByUid(u.getId())) {
            if (!lrid.contains(j)) {
```

```
                for (UserRole userRole : urd.findBy2Properties
                ("role.id", j,
                        "userinfo.id", u.getId())) {
                    urd.delete(userRole);
                }
            }
        }
        for (Integer i : lrid) {                          //进行遍历
            if (urd.findRoleIdByUid(u.getId()).contains(i)) {
                continue;
            } else {
                ur = new UserRole();
                ur.setUserinfo(u);
                r.setId(i);
                ur.setRole(r);
                urd.save(ur);
            }
        }
        return "success";
    }
    public void removeUser(Userinfo u) {          //删除用户
        for (UserRole ur : urd.findByProperty("userinfo.id", u.getId())) {
            urd.delete(ur);
        }
        ud.delete(u);
    }
    public Userinfo findSingleUser(Userinfo u) {//单查用户
        return ud.findById(u.getId());
    }
    public List<Integer> getRole(Userinfo u) { //查看用户所属角色
        if (u.getId() == null) {
            u = ud.findByUsername(u.getUsername()).get(0);
        }
        return urd.findRoleIdByUid(u.getId());
    }
    public String login(Userinfo u) {             //用户登录判断用户名和密码
        if (ud.findByUsername(u.getUsername()).size() != 0
                && ud.findByUsername(u.getUsername()).g
                et(0).getPassword()
                        .equals(u.getPassword())) {
            return "success";
        }
        return "input";
    }
    public void modifyUser(Userinfo u) {          //修改用户
        ud.update(u);
    }
    public String newUser(Userinfo u, List<Integer> lrid) {//新增用户
        if (lrid.size() == 0) {                       //对 lrid 的大小进行判断
            return "error";
        }
        if (ud.findByUsername(u.getUsername()).size() == 0) {
            ud.save(u);                               //调用 save()方法
            for (Integer i : lrid) {
                UserRole ur = new UserRole();
                Role r = new Role();
                r.setId(i);
                ur.setUserinfo(u);
                ur.setRole(r);
                urd.save(ur);
```

```
            }
            return "success";
        }
        return "input";
    }
    public List<Userinfo> operateUser() {           //查看所有用户
        return ud.findAll();
    }
    public String regist(Userinfo u) {              //用户注册判断用户名
        if (ud.findByUsername(u.getUsername()).size() == 0) {//判断大小
            ud.save(u);                             //调用 save()方法
            UserRole ur = new UserRole();           //创建 UserRole 对象
            Role r = new Role();                    //创建 Role 对象
            //判断用户名
            ur.setUserinfo(u);
            r.setId(2);
            ur.setRole(r);
            urd.save(ur);
            return "success";
        }
        return "input";
    }
}
```

在 applicationContext.xml 文件中配置 UserinfoFacade 类。

```
    <!--对 UserinfoFacade 类进行配置 -->
    <bean id="userinfoFacade" class="com.cjg.user.service.
    UserinfoFacade">
        <property name="urd" ref="userRoleDAO" />
        <property name="ud" ref="userinfoDAO" />
        <property name="rd" ref="roleDAO" />
    </bean>
```

注意：在上述代码中，分别实现了 IUserinfoFacade 接口中的所有方法。

26.7.3　用户操作的表现层

在设计实现关于 userinfo 表操作的表现层时，利用 Struts 2.0 框架实现页面的跳转。操作 userinfo 表涉及的页面有：实现查看全部角色信息的页面 operateRole.jsp、实现对角色授权的页面 authorization.jsp、实现新增角色的页面 newRole.jsp，以及实现单查角色的页面 listSingleRole.jsp。

角色操作的所有请求都由 Struts 2.0 框架中名为 RoleAction 的 Action 类来处理，RoleAction.java 的具体内容如代码 26.47 所示。

代码 26.47　关于用户操作的跳转：UserinfoAction.java

```
...
public class UserinfoAction {
    // 针对于页面的成员变量
    private Userinfo u;
    private List<Userinfo> lu;
    private List<Integer> lrid;
```

```
    private List<Role> lr;
    private IUserinfoFacade uf;
    private IRoleFacade rf;
    public UserinfoAction() {                        //构造函数
        lrid = new ArrayList<Integer>();
        u = new Userinfo();
    }
    //省略属性 rf、u、lu、lrid、lr 和 uf 的 set()和 get()方法
...
    //编写处理 login()请求方法
    public String login() {
        if (uf.login(u).equals("success")) {
            HttpSession hs = ServletActionContext.getRequest().
            getSession();
            hs.setAttribute("role", uf.getRole(u));
        }
        return uf.login(u);
    }
    public String regist() {                //编写处理 regist()请求方法
        return uf.regist(u);
    }
    public String operateUser() {           //编写处理 operateUser()请求方法
        lu = uf.operateUser();
        return Action.SUCCESS;
    }
    public String findSingleUser() {        //编写处理 findSingleUser()请求方法
        u = uf.findSingleUser(u);
        return Action.SUCCESS;
    }
    public String modifyUser() {            //编写处理 modifyUser()请求方法
        uf.modifyUser(u);
        return Action.SUCCESS;
    }
    public String operateUR() {             //编写处理 operateUR()请求方法
        lr = rf.operateRole();
        return Action.SUCCESS;
    }
    public String newUser() {               //编写处理 newUser()请求方法
        return uf.newUser(u, lrid);
    }
    public String removeUser() {            //编写处理 deleteUser()请求方法
        uf.removeUser(u);
        return Action.SUCCESS;
    }
    public String listRole() {              //编写处理 listRole()请求方法
        ServletActionContext.getRequest().setAttribute("had",
        uf.getRole(u));
        HttpSession hs = ServletActionContext.getRequest().getSession();
        if (u != null) {
            hs.setAttribute("user", u);
        }
        lr = rf.operateRole();
        return Action.SUCCESS;
    }
    public String changeUR() {              //编写处理 changeUR()请求方法
        HttpSession hs = ServletActionContext.getRequest().getSession();
```

```
        return uf.changeUR(lrid, (Userinfo) hs.getAttribute("user"));
    }
    public String exit() {                //编写处理 exit()请求方法
        HttpSession hs = ServletActionContext.getRequest().getSession();
        hs.removeAttribute("role");
        return Action.SUCCESS;
    }
}
```

首先在 applicationContext.xml 文件中配置 UserinfoAction 类。

```
    <!--对 UserinfoAction 类进行配置 -->
    <bean id="userinfoAction" scope="prototype" class="com.cjg.user.
    action.UserinfoAction">
        <property name="rf" ref="roleFacade" />
        <property name="uf" ref="userinfoFacade" />
    </bean>
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
        <!-- 配置关于 operateUser 的 Action-->
        <action name="operateUser" method="operateUser" class=
        "userinfoAction">
            <result>/page/user/listUser.jsp</result>
        </action>
        <!-- 配置关于 findSingleUser 的 Action-->
        <action name="findSingleUser" method="findSingleUser" class=
        "userinfoAction">
            <result>/page/user/listSingleUser.jsp</result>
        </action>
        <!-- 配置关于 modifyUser 的 Action-->
        <action name="modifyUser" method="modifyUser" class="userinfo-
        Action">
            <result type="chain">operateUser</result>
        </action>
        <!-- 配置关于 newUser 的 Action-->
        <action name="newUser" method="newUser" class="userinfoAction">
            <result type="chain">operateUser</result>
            <result name="input">/page/user/userHad.jsp</result>
            <result name="error">/page/user/null.jsp</result>
        </action>
        <!-- 配置关于 operateUR 的 Action-->
        <action name="operateUR" method="operateUR" class=
        "userinfoAction">
            <result>/page/user/newUser.jsp</result>
        </action>
        <!-- 配置关于 removeUser 的 Action-->
        <action name="removeUser" method="removeUser" class=
        "userinfoAction">
            <result type="chain">operateUser</result>
        </action>
        <!-- 配置关于 listRole 的 Action-->
        <action name="listRole" method="listRole" class="userinfoAction">
            <result>/page/user/listRole.jsp</result>
        </action>
        <!-- 配置关于 changeUR 的 Action-->
        <action name="changeUR" method="changeUR" class="userinfoAction">
            <result type="chain">operateUser</result>
            <result name="input">/page/user/null.jsp</result>
```

```
        </action>
        <!-- 配置关于 exit 的 Action-->
        <action name="exit" method="exit" class="userinfoAction">
            <result>/index.jsp</result>
        </action>
```

1. 关于用户登录的页面login.jsp

页面 login.jsp 用来实现用户的登录功能，该页面的具体内容如代码 26.48 所示。

代码 26.48　用户登录页面：login.jsp

```
...
    <body>
        <table>
            <!--用户名输入框-->
            <td >用户名:
            <s:textfield name="u.username" theme="simple" />
            </td>
            <!--密码输入框-->
            <td>密码:
            <s:password name="u.password" theme="simple" />
            </td>
            <!--提交和重置按钮->
            <s:submit value="登录" theme="simple" />
            <s:reset value="重置" theme="simple" />
        </table>
    </body>
...
```

2. 关于用户注册的页面regist.jsp

页面 regist.jsp 用来实现用户的注册功能，该页面的具体内容如代码 26.49 所示。

代码 26.49　用户注册页面：regist.jsp

```
...
    <body>
        <form id="form" method="post" name="regist" action=
        "regist.action">
            <table >
                <!--用户名输入框-->
                <td>用户名:
                <s:textfield name="u.username" theme="simple" />
                </td>
                <!--用户密码输入框-->
                <td>用户密码:
                <s:password name="u.password" theme="simple" />
                <!--提交和重置按钮-->
                <s:submit value="注册" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </form>
    </body>
...
```

3．关于新增角色的页面listUser.jsp

页面 listUser.jsp 用来实现新增角色页，该页面的具体内容如代码 26.50 所示。

代码 26.50　新增角色页面：listUser.jsp

```
...
    <body>
        <form action="modifyUser.action" method="post">
            <table>
                <!--表格头标题-->
                <TD >用户名</TD >
                <TD >密码</TD >
                <TD >删除操作</TD >
                <TD >角色操作</TD >
                <!--遍历角色结果-->
                <s:iterator value="lu">
                <!--输出用户名-->
                <a href='<s:url action="findSingleUser"><s:param name=
                "u.id" value="id" />
                    </s:url>' target="content"> <s:property value=
                    "username" /> </a>
                <!--输出密码-->
                <s:property value="password" />
                <!--实现删除功能-->
                <a href='<s:url action="removeUser"><s:param name="u.id"
                value="id" />
                    </s:url>' target="content">删除</a>
                <!--实现设置角色功能-->
                <a href='<s:url action="listRole"><s:param name="u.id"
                value="id" />
                    </s:url>' target="content">设置角色</a>
                </s:iterator>
                <!--关于添加用户的超级链接-->
                <a href="operateUR.action" target="content">添加用户</a>
            </table>
        </form>
    </body>
...
```

4．关于添加用户的页面newUser.jsp

页面 newUser.jsp 用来实现添加用户的功能，该页面的具体内容如代码 26.51 所示。

代码 26.51　添加用户页面：newUser.jsp

```
...
    <body>
        <form action="modifyUser.action" method="post">
            <table>
                <!--用户名输入框-->
                <TD >用户名
                <s:textfield name="u.username" theme="simple" />
                </TD >
                <!--用户密码输入框-->
```

```
                <TD >用户密码
                <s:password name="u.password" theme="simple" />
                </TD >
                <!--用户角色选择框->
                <TD >用户角色
                <s:checkboxlist name="lrid" theme="simple" list="lr"
                listKey="id"
                                listValue="rolename">
                </s:checkboxlist>
                </TD >
                <!--提交和重置按钮-->
                <s:submit value="新增用户" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </form>
    </body>
...
```

5. 关于修改用户信息的页面listSingleUser.jsp

页面 listSingleUser.jsp 用来实现对用户进行修改功能，该页面的具体内容如代码 26.52 所示。

代码 26.52 修改用户信息页面：listSingleUser.jsp

```
...
    <body>
        <form action="modifyUser.action" method="post">
            <table>
                <!--用户序号输入框-->
                <TD >用户序号
                <s:textfield name="u.id" value="%{u.id}" theme="simple" />
                </TD >
                <!--用户名输入框-->
                <TD >用户名
                <s:textfield name="u.username" value="%{u.username}"
                theme="simple" />
                </TD >
                <!--用户密码输入框-->
                <TD >用户密码
                <s:textfield name="u.password" value="%{u.password}"
                theme="simple" />
                </TD >
                <!--提交和重置按钮-->
                <TD >用户密码
                <s:textfield name="u.password" value="%{u.password}"
                theme="simple" />
                </TD >
                <!--提交和重置按钮-->
                <s:submit value="修改用户" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </form>
    </body>
...
```

6．设计修改用户角色的页面listRole.jsp

页面 listRole.jsp 用来实现对角色进行授权功能，该页面的具体内容如代码 26.53 所示。

代码 26.53　修改用户角色页面：listRole.jsp

```
...
    <body>
        <form action="changeUR.action" method="post">
            <table>
                <!--遍历角色-->
                <s:iterator value="lr">
                    <TD >
                        <s:if test="id in #request.had">
                        <!--角色 ID 选择框-->
                        <s:checkbox theme="simple" name="lrid"
                            fieldValue="%{id}" value="true" labelposition=
                            "left">
                        <s:property value="rolename" />
                        </s:checkbox>
                        </s:if>
                        <s:else>
                        <!--角色名称选择框-->
                        <s:checkbox theme="simple" name="lrid"
                            fieldValue="%{id}" value="false" labelposition=
                            "left">
                        <s:property value="rolename" />
                        </s:checkbox>
                        </s:else>
                    </TD>
                </s:iterator>
                <!--提交和重置按钮-->
                <s:submit value="改变用户角色" theme="simple" />
                <s:reset value="重置" theme="simple" />
            </table>
        </form>
    </body>
...
```

7．设计登录失败的页面err.jsp

页面 err.jsp 为用户登录失败的页面，该页面的具体内容如代码 26.54 所示。

代码 26.54　登录失败页面：err.jsp

```
...
    <body>
...
            <TD>
                <!--链接-->
                <a href="login.jsp">用户名或密码错误，点击这里返回登录页面</a>
            </TD>
...
```

```
</body>
...
```

8. 设计注册失败的页面had.jsp

页面 had.jsp 为用户注册失败的页面，该页面的具体内容如代码 26.55 所示。

代码 26.55　注册失败页面：had.jsp

```
...
    <body>
...
            <TD>
                <!--链接-->
                <a href="regist.jsp">该用户名已存在，点击这里重新注册</a>
            </TD>
...
    </body>
...
```

9. 设计添加用户失败的页面userHad.jsp

页面 userHad.jsp 为添加用户失败的页面，该页面的具体内容如代码 26.56 所示。

代码 26.56　添加用户失败页面：userHad.jsp

```
...
    <body>
...
            <TD>
                <!--链接-->
                <a href="operateUR.action">该用户名已存在，点击这里重新添加</a>
            </TD>
...
    </body>
...
```

10. 设计修改用户角色失败的页面null.jsp

页面 null.jsp 为修改用户角色失败的页面，该页面的具体内容如代码 26.57 所示。

代码 26.57　修改用户角色失败页面：null.jsp

```
...
    <body>
...
            <!--链接-->
            <TD>
                <a href="operateUser.action">每个用户要有一个角色，
                点击这里返回</a>
            </TD>
...
    </body>
...
```

26.8　小　　结

本章主要介绍一个完整的权限管理系统，该系统实现了经典的“用户——角色——功能模块”模式，基于 Struts 2.x+Spring+JPA 框架构建而成。在具体实现权限管理系统时，从 Java EE 开发标准的 4 层结构体系详细讲解了该系统的各个模块：模块操作、功能操作、角色操作和用户操作。其中 Struts 2.x 框架实现表示层页面的跳转，JPA 框架实现由数据库记录转变成 POJO 对象的持久层，Spring 框架主要实现该系统业务逻辑的服务层。

第 27 章　商业银行设备巡检系统（Struts 2.x+Spring+Hibernate）

通过第 26 章的实战，已经知道了如何开发复杂的商业银行设备巡检系统。本章将在第 26 章权限管理系统的基础上实现商业银行设备巡检系统。

由于权限管理系统是基于 Struts 2.x+Spring+JPA 框架，所以本章将通过 Struts 2.x+Spring+Hibernate 框架来实现商业银行设备巡检系统。由于商业银行设备巡检系统比较复杂，所以在开发该系统时先按照面向各个应用模块分层，然后再对各个模块按照 MVC 分层。

27.1　商业银行设备巡检系统概述

在开发商业银行设备巡检系统之前，首先要了解需要实现什么功能，如何实现这些功能等。在本节中将对商业银行设备巡检系统的需求、业务流程进行细致地分析和讲解。

27.1.1　需求分析

随着网络应用的日益普及，面向日常运作和管理的方式也发生了很大的变化。随着组织规模的不断扩大，商业银行越来越希望能够打破时间、地域的限制，提高整个组织的运行效率。本章的商业银行设备巡检系统正是基于这种需求而诞生的。

本章的商业银行设备巡检系统分为系统管理、设备巡检和设备报修 3 大模块。通过系统管理可以为超级用户、巡检中心员工、银行员工和巡检工赋予相应的权限以及设置一些必要信息。通过设备巡检和设备报修可以考核各种员工的绩效。

本系统各个模块基本功能如下所述。

- 系统管理模块：岗位管理、日记管理、用户管理、银行设备种类管理、部门管理、巡检工管理、巡检组管理、银行网点管理和设备问题报修管理。
- 设备巡检管理模块：设备巡检和网点对巡检确认。
- 设备报修管理模块：巡检中心查看报修信息、网点查看报修信息、网点设备报修、巡检中心分配小组、值班员报修、维修工确认维修及网点对报修确认。

27.1.2　业务分析——系统管理关于超级管理用户操作

商业银行设备巡检系统由 3 大部分组成，系统管理、设备巡检和设备报修。下面将演

示系统管理中关于超级管理用户的操作。

1. 岗位管理

在该系统中由于用户处于不同的岗位，所以会拥有不同的功能。在商业银行中需要对 3 种岗位的员工进行考核，分别为巡检中心、巡检组和银行。由于每个系统都需要一个超级用户，所以还需要设定一个高级管理员岗位。

单击“系统管理”|“岗位管理”按钮就可以打开岗位管理页面（如图 27.1 所示），在该页面中可以增加岗位、修改岗位和配置岗位。

注意：名称为巡检组 1 和巡检组 2 的岗位为巡检组岗位，名 bank1 和 bank2 的岗位为银行岗位，名为巡检中心 1 的岗位为巡检中心岗位，最后高级管理员则为高级管理员岗位。

（1）如果想创建一个新的岗位，可以单击图 27.1 中的“增加”按钮打开创建岗位的页面（如图 27.2 所示），按照该页面的要求填写相应的信息就可以生成相应新岗位。



图 27.1　岗位管理页面



图 27.2　创建岗位页面

注意：所有岗位中，除了巡检组外其他的岗位都为管理性质。

（2）如果想修改已经存在的岗位，可以单击岗位记录里“操作”中的“编辑”按钮。单击“编辑”按钮可以打开修改岗位信息页面（如图 27.3 所示），修改完单击“修改”按钮就可以实现修改岗位的功能。

（3）如果想配置岗位，即为岗位赋予特定的功能，可以单击岗位记录里“操作”中的“模块管理”按钮。单击“模块管理”按钮可以打开程序功能列表页面（如图 27.4 所示），在该页面中会显示出该系统的 3 大模块：系统管理、报修管理和巡检管理。如果想把系统管理模块中的某个功能赋予该岗位，可以单击模块记录里“操作”中的“操作”按钮。例如单击系统管理记录的“操作”按钮可以打开系统管理页面列表页面（如图 27.5 所示），

在该页面中就可以选择任意一个功能赋予岗位。



图 27.3　修改岗位信息页面



图 27.4　程序功能列表页面



图 27.5　系统管理页面列表页面

2. 部门管理

在该系统中用户不仅需要通过岗位获取相应的功能外，而且还属于不同的部门。如果想创建新的部门，可以单击“系统管理”|“部门管理”按钮就可以打开部门管理页面（如图 27.6 所示），在该页面中可以增加部门、修改部门及删除部门。

注意：之所以有些部门记录里“操作”中有“删除”按钮，而有些却没有，这是因为当部门中没有用户时，则会出现“删除”按钮；当部门中还存在用户时，则不会出现“删除”按钮。

（1）如果想创建一个新部门，可以单击“增加”按钮打开新增部门的页面（如图 27.7 所示），按照该页面的要求填写相应的信息就可以生成相应的新部门。

（2）如果想修改已经存在的部门，可以单击部门记录里“操作”中的“编辑”按钮。单击“编辑”按钮可以打开修改部门页面（如图 27.8 所示），修改完单击“修改”按钮就可以实现修改部门的功能。

（3）如果想删除没有用户的部门，可以单击部门记录里“操作”中的“删除”按钮，实现删除该部门的功能。



图 27.6　部门管理页面



图 27.7　创建部门页面



图 27.8　修改部门信息

3．用户管理

当创建完岗位和部门后，就可以通过“系统管理”|“用户管理”按钮来创建各种用户。单击“系统管理”|“用户管理”按钮就可以打开用户管理页面（如图 27.9 所示），在该页面中可以进行增加用户、修改用户、删除用户、修改用户状态和查询用户的操作。



图 27.9　用户管理页面

（1）如果想创建一个新的用户，可以单击“增加”按钮打开创建用户的页面（如图 27.10 所示），按照该页面的要求填写和选择相应的信息就可以生成相应的新用户。

系统管理>>用户管理>>用户新增

用户登陆ID	
用户登陆密码	
确认密码	
用户中文名称	
用户所属部门	巡检中心
用户所在岗位	巡检中心1
用户状态	⊙启用 ○禁用
	新增 取消

图 27.10　创建用户页面

（2）如果想修改已经存在的用户，可以单击部门记录里“操作”中的“编辑”按钮。单击“编辑”按钮可以打开用户修改页面（如图 27.11 所示），修改完单击“修改”按钮就可以实现修改用户的功能。

系统管理>>用户管理>>用户修改

用户修改

用户登陆ID	admin
用户登陆密码	••••••
确认密码	••••••
用户中文名称	高级管理员
用户所属部门	董事会
用户所在岗位	高级管理员
用户状态	⊙启用 ○禁用
	修改 返回

图 27.11　修改岗位信息

（3）如果想删除用户，可以单击用户记录里“操作”中的“删除”按钮实现删除该用户的功能。用户存在两种状态，如果用户是启用状态，则该用户可以登录该系统，如果用户是禁用状态，则该用户不能登录该系统。单击用户记录里“操作”中的“更该用户状态”按钮可实现更改用户状态的功能。

（4）当用户记录太多时，如果想查看某个用户记录，则会非常麻烦。在该系统中可以通过单击“查询”按钮打开查询页面（如图 27.12 所示）。在该页面中填写相应的信息，单击“查询”按钮就可以转到相应的查询结果页面（如图 27.13 所示）。

系统管理>>用户管理

新增　查询

用户登陆ID：11　Search

用户登陆ID	用户中文名字	用户所在部门	用户所属岗位	用户状态	操作
admin	高级管理员	董事会	高级管理员	启用	
22xjg	巡检工	巡检工	巡检组2	启用	
22bank	bank	bank	bank2	启用	
11zx	巡检中心	巡检中心	巡检中心1	启用	
11xjg	巡检工	巡检工	巡检组1	启用	
11bank	bank	bank	bank1	启用	

共6 项　每页10项　当前第1 页　共1 页　第　页　跳转

图 27.12　查询页面



图 27.13　查询结果页面

注意：当单击“查询”按钮时，则调用相应的方法实现模糊查询功能。

4．银行网点管理

超级用户除了要设置上述的基本信息外，还必须要通过“系统管理”|“银行网点管理”按钮来设置各个银行的基础信息。单击“系统管理”|“银行网点管理”按钮就可以打开银行网点管理页面（如图 27.14 所示），在该页面中可以创建新银行网点。



图 27.14　银行网点管理页面

如果想创建一个新的银行网点，可以单击“增加”按钮打开创建银行的页面（如图 27.15 所示），按照该页面的要求填写并选择相应的信息就可以生成相应的新银行网点。



图 27.15　创建银行网点页面

27.1.3　业务分析——系统管理关于银行员工操作

由于银行用户与超级用户（admin）所处的岗位不同，所以这两个用户拥有各个模块中的功能也不尽相同。当银行用户（11bank）登录该系统时，系统管理模块页面如图 27.16

所示。



图 27.16　银行用户系统管理模块

🔔注意：当银行用户登录该系统时，必须在自己所属于银行里的计算机上登录。因为该系统与银行的 IP 号绑定。如果通过其他地方的计算机登录时，则各个功能不能被使用。

1．设备问题报修管理

设备问题报修管理主要用来创建银行设备常见的问题，如果想创建一个新的设备问题，可以先单击“设备问题报修管理”按钮就可以打开设备报修问题列表页面（如图 27.17 所示）。在该页面中可以增加设备报修问题、修改设备报修问题和删除设备报修问题的功能。



图 27.17　设备报修问题列表

（1）如果想创建一个新的设备报修问题，可以单击“增加”按钮打开新增设备报修问题页面（如图 27.18 所示）。在该页面中填写相应的内容后，单击“保存”按钮就可以实现创建新设备报修问题。



图 27.18　增加设备问题页面

（2）如果想修改已经存在的设备报修问题，可以单击部门记录“操作”中的“编辑”按钮。单击“编辑”按钮可以打开修改设备报修问题页面（如图 27.19 所示），修改完单击“提交”按钮，就可以实现修改该设备报修问题的功能。

图 27.19　修改报修问题

（3）如果想删除用户，可以单击用户记录里“操作”中的“删除”按钮，实现删除该设备报修问题的功能。

2. 银行设备种类管理

银行设备种类管理主要用来创建银行设备的类型，如果想创建一个新的设备类型，可以先单击“银行设备种类管理”按钮打开银行设备种类管理列表页面（如图 27.20 所示）。在该页面中可以实现增加银行设备种类、修改银行设备种类和删除银行设备种类的功能。

图 27.20　银行设备种类管理列表

（1）如果想创建一个新的银行设备种类，可以单击“增加”按钮打开新增设备种类问题页面（如图 27.21 所示），在该页面中填写相应的内容后单击“保存”按钮，就可以实现创建新设备的种类。

图 27.21　新增设备种类问题页面

（2）如果想修改已经存在的银行设备种类，可以单击银行设备种类记录里“操作”中的“编辑”按钮。单击“编辑”按钮可以打开修改银行设备种类管理页面（如图 27.22 所示），修改完单击“修改”按钮就可以实现修改银行设备种类的功能。

系统管理>>银行设备种类管理列表>>修改银行设备种类管理

设备种类ID	2
设备名称	显示器

修改　返回

图 27.22　修改岗位信息

（3）如果想删除银行设备种类，可以单击银行设备种类记录里“操作”中的“删除”按钮，实现删除该银行设备种类的功能。

3．银行设备明细管理

所谓银行设备明细管理是指对银行的所有设备进行管理，即为每个设备创建一个正常与否的记录。如果想实现该功能，可以通过“系统管理”|“银行网点管理”选项打开银行网点管理页面（如图 27.23 所示）后，然后通过单击“新增银行明细”按钮来实现。在银行网点列表页面可以实现编辑银行网点、查看银行设备明细和新增银行明细的功能。

系统管理>>银行网点列表

新增

银行ID	银行名称	银行位置经度	银行位置纬度	银行IP	操作
2	bank2	20.0	20.0	192.168.1.40	查看银行设备明细　新增银行明细
1	bank1	20.0	20.0	192.168.1.55	查看银行设备明细　新增银行明细

共2 项　每页　10项　当前第1 页　共1 页　跳转

图 27.23　银行网点列表

（1）如果想修改已经存在的银行网点，可以单击银行网点记录里“操作”中的“编辑”按钮。单击“编辑”按钮可以打开修改银行网点页面（如图 27.24 所示），修改完单击“修改”按钮就可以实现修改用户的功能。

系统管理>>银行网点管理>>修改银行网点

银行编号	2
银行名称	银行2
银行位置经度	20.0
银行位置纬度	20.0
银行IP	192.168.1.52

修改　取消

图 27.24　修改银行网点信息

（2）如果想查看已经存在的银行网点，可以单击银行网点记录里“操作”中的“查看设备明细”按钮。单击“查看设备明细”按钮可以打开银行设备明细列表页面（如图 27.25 所示），在该页面中可以实现新增银行设备明细、修改银行设备明细和删除银行设备明细功能。

（3）如果想创建一个新银行设备明细，可以单击“增加”按钮打开创建银行设备明细

的页面（如图 27.26 所示），按照该页面的要求填写和选择相应的信息就可以生成相应的新银行设备明细。



系统管理>>银行设备明细列表

新增

设备流水ID	银行编号ID	设备名类ID	购入价格	购入时间	设备状态	设备折旧残值	操作
2x2	银行2	显示器	85.0	2009-06-12	设备正常	12.0	
2x1	银行2	显示器	87.0	2009-06-10	设备正常	12.0	
2p2	银行2	盘	54.0	2009-06-19	设备正常	12.0	
2p1	银行2	盘	54.0	2009-06-18	设备正常	12.0	

共4项 每页 10项 当前第1页 共1页 共1页 跳转

图 27.25　查看银行设备明细



图 27.26　新增银行设备明细页面

（4）如果想修改已经存在的银行设备明细，可以单击银行设备明细记录里“操作”中的“编辑”按钮。单击“编辑”按钮可以打开修改银行设备明细页面（如图 27.27 所示），修改完单击“修改”按钮就可以实现修改用户的功能。



图 27.27　修改银行设备明细信息

（5）如果想删除银行设备明细，可以单击银行设备明细记录里“操作”中的“删除”按钮，实现删除该银行设备明细的功能。

27.1.4　业务分析——系统管理关于巡检工操作

商业银行设备巡检系统由 3 大部分组成，系统管理、设备巡检和设备报修。下面将演示系统管理中关于巡检工的操作。

1. 创建巡检工

如果想创建巡检工，可以通过“系统管理”|“巡检工管理”按钮来创建巡检工。单击“系统管理”|“巡检工管理”按钮可以打开巡检工管理页面（如图 27.28 所示），在该页面中可以增加巡检工、修改巡检工、删除巡检工的功能。



图 27.28　巡检工管理页面

（1）如果想创建一个新的巡检工，可以单击“增加”按钮打开创建巡检工的页面（如图 27.29 所示），按照该页面的要求填写和选择相应的信息就可以生成相应的新巡检工。



图 27.29　创建巡检工页面

（2）如果想修改已经存在的巡检工，可以单击巡检工记录里“操作”中的“编辑”按钮。单击“编辑”按钮可以打开修改巡检工页面（如图 27.30 所示），修改完后单击“提交”按钮就可以实现修改巡检工的功能。



图 27.30　修改巡检工

（3）如果想删除巡检工，可以单击巡检工记录里“操作”中的“删除”按钮，实现删除该巡检工的功能。

2. 创建巡检组

如果想创建巡检组，可以通过“系统管理”|“巡检组管理”按钮来创建巡检组。单击“系统管理”|“巡检组管理”按钮就可以打开巡检组管理页面（如图 27.31 所示），在该页面中可以增加巡检组、修改巡检组、配置巡检组和删除巡检工的功能。

（1）如果想创建一个新的巡检组，可以单击“增加”按钮打开创建巡检组的页面（如

图 27.32 所示），按照该页面的要求填写及选择相应的信息就可以生成相应的新巡检组。



图 27.31　巡检组管理页面



图 27.32　创建组页面

（2）如果想修改已经存在的巡检组，可以单击巡检工记录里“操作”中的“编辑”按钮。单击“编辑”按钮可以打开修改巡检组页面（如图 27.33 所示），修改完单击“提交”按钮就可以实现修改巡检组的功能。



图 27.33　修改巡检组

（3）如果想重新分配巡检组中的巡检工，可以单击巡检组记录里“操作”中的“重新分配”按钮，打开重新分配页面（如图 27.34 所示），在该页面中选择新的巡检工就可以完成该功能。

注意：巡检工是通过其所属的巡检组的用户名和密码来实现登录该系统。配置巡检组功能就是实现巡检工与巡检组之间的绑定。

（4）如果想查看巡检组信息，可以单击巡检组记录里“操作”中的“查看”按钮，打

开查看巡检组信息页面（如图 27.35 所示）。



图 27.34　重新分配页面



图 27.35　查看巡检组信息

27.1.5　业务分析——系统管理关于设备巡检

商业银行设备巡检系统由 3 大部分组成，系统管理、设备巡检和设备报修。下面将演示如何实现设备巡检，具体过程如下。

（1）巡检工抵达银行后，首先要巡检完该银行的所有设备，然后通过银行的计算机登录该系统。通过单击“巡检管理”|“设备巡检”按钮，在出现的巡检列表页面（如图 27.36 所示）中添加相应的巡检记录。



图 27.36　巡检工设备巡检

注意：巡检管理模块功能是与银行的 IP 绑定，如果巡检工在其他地方登录，则不会出现上述页面。

在巡检列表页面中，如果想增加相应的巡检记录，可以通过单击该页面的“增加”按钮打开新增巡检记录页面（如图 27.37 所示），在该页面中填写相应的内容则可以完成该功能。

设备巡检管理>>设备巡检>>新增巡检记录

巡检组Id	9
设备状态	正常

提交　取消

图 27.37　新增设备巡检

（2）巡检工退出该系统后，该银行相应人员还应该登录该系统对巡检工的巡检工作进行确认。通过单击“巡检管理”|“网点对巡检确认”按钮就可以打开网点确认页面（如图 27.38 所示）。

中国移动通信 CHINA MOBILE　系统管理　报障管理　巡检管理　退出

网点对巡检确认

设备巡检管理>>网点确认

巡检序列号	设备状态	巡检组Id	巡检时间	巡检确认	操作
160	正常	1zu	2009-06-13 14:43:25	结束巡检	
111	异常	1zu	2009-06-08 16:26:15	结束巡检	
49	异常	1zu	2009-06-08 15:20:43	结束巡检	
36	异常	1zu	2009-06-08 15:00:35	银行已确认	
28	正常	1zu	2009-06-08 00:00:00	银行已确认	

共5 项　每页 10 项　当前第 1 页　共 1 页　页 跳转

图 27.38　网点确认页面

如果想确认巡检记录，可以单击巡检记录里“操作”中的“银行确认”按钮。单击“银行确认”按钮就可以实现对该记录的确认，确认后的页面如图 27.39 所示。

设备巡检管理>>网点确认

巡检序列号	设备状态	巡检组Id	巡检时间	巡检确认	操作
160	正常	1zu	2009-06-13 14:43:25	银行已确认	
111	异常	1zu	2009-06-08 16:26:15	结束巡检	
49	异常	1zu	2009-06-08 15:20:43	结束巡检	
36	异常	1zu	2009-06-08 15:00:35	银行已确认	
28	正常	1zu	2009-06-08 00:00:00	银行已确认	

共5 项　每页 10 项　当前第 1 页　共 1 页　页 跳转

图 27.39　确认后记录

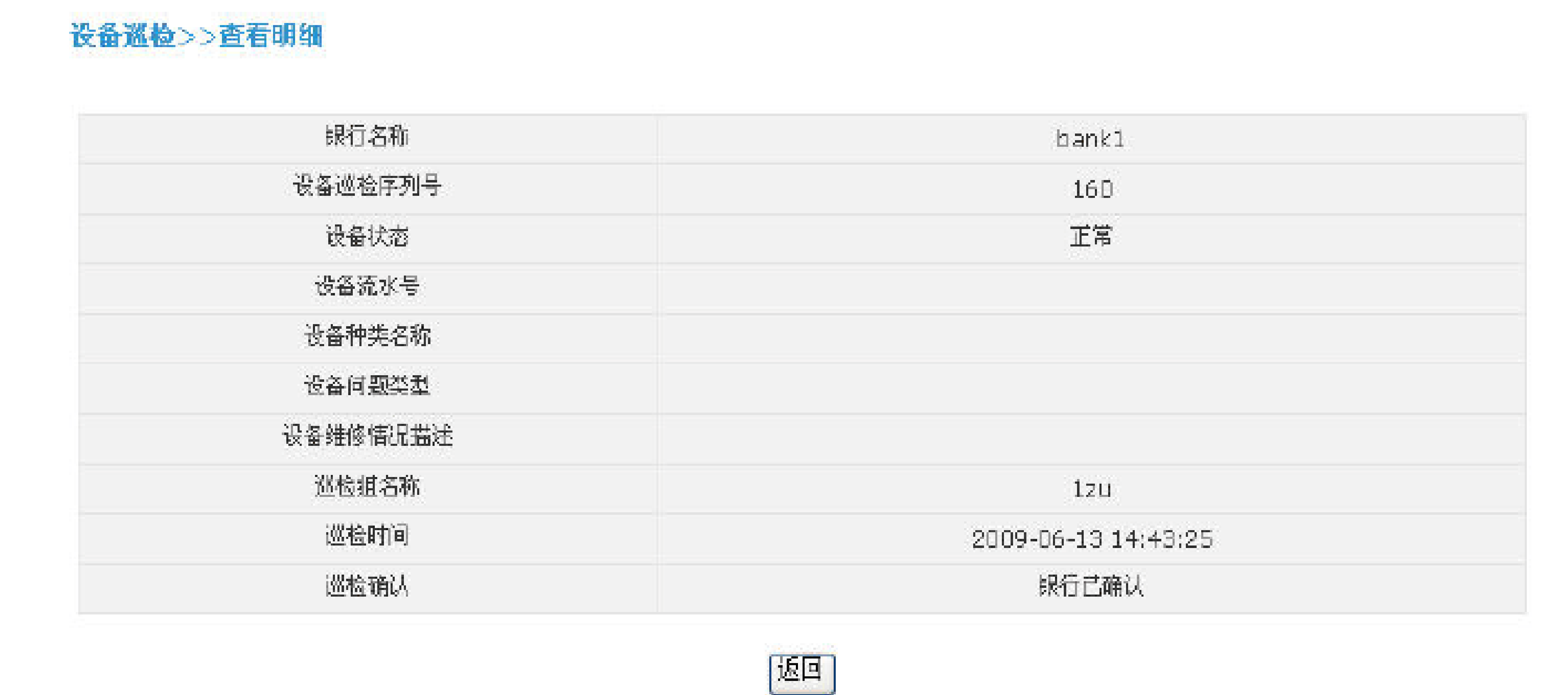
（3）如果想对巡检工与银行员工进行绩效考核，巡检中心的领导登录该系统后，通过“巡检管理”|“设备巡检”按钮就可以打开巡检列表（如图 27.40 所示）。如果想查看具体某记录的信息，可以单击巡检记录里“操作”中的“查看”按钮，打开关于该记录的巡检明细页面，如图 27.41 所示。



巡检序列号	银行	巡检时间	巡检组Id	设备状态	巡检确认	操作
160	bank1	2009-06-13 14:43:25	9	正常	银行已确认	
111	bank1	2009-06-08 16:26:15	9	异常	结束巡检	
49	bank1	2009-06-08 15:20:43	9	异常	结束巡检	
36	bank1	2009-06-08 15:00:35	9	异常	银行已确认	
28	bank1	2009-06-08 00:00:00	9	正常	银行已确认	

共5项 每页 10项 当前第 1页 共 1页

图 27.40　巡检列表


设备巡检>>查看明细

银行名称	bank1
设备巡检序列号	160
设备状态	正常
设备流水号	
设备种类名称	
设备问题类型	
设备维修情况描述	
巡检组名称	1zu
巡检时间	2009-06-13 14:43:25
巡检确认	银行已确认

返回

图 27.41　查看巡检明细

27.1.6　业务分析——设备报修中关于银行报修

商业银行设备巡检系统由 3 大部分组成，系统管理、设备巡检和设备报修。下面将演示如何实现设备报修中的银行报修，具体过程如下：

（1）银行员工发现有坏的设备时，可以通过登录该系统进行设备报修。通过单击“巡检管理”|“网点设备报修”按钮就可以打开网点设备报修页面（如图 27.42 所示），在该页面中选择相应的选项就可以完成该功能。



图 27.42　网点设备报修页面

（2）当网点设备报修后，还需要银行员工确认。通过单击“巡检管理”|“网点对报修确认”按钮就可以打开网点对报修确认页面（如图 27.43 所示），在该页面中单击报修记录里“操作”中的“确认维修”按钮。该报修的记录状态就由结束报修变成正在申请

状态。



图 27.43　确认报修

（3）当银行确认完报修记录后，巡检中心的工作员工就需要登录该系统，对该条报修记录进行小组分配。通过“报修管理”|“巡检中心分配小组”按钮打开分配小组页面（如图 27.44 所示），单击报修记录里“操作”中的“分配小组”按钮，就可以打开分配小组页面（如图 27.45 所示）。



图 27.44　分配小组



图 27.45　实现分配小组

（4）当巡检中心为报修记录分配完小组后，通过“巡检中心查看报修信息”按钮打开设备报修记录列表（如图 27.46 所示），就可以发现该记录里“小组分配状态”已经变成“已分配小组”。



图 27.46　查看设备报修记录

27.1.7　业务分析——设备报修中关于巡检工维修

商业银行设备巡检系统由 3 大部分组成，系统管理、设备巡检和设备报修。下面将演示如何实现设备报修中的巡检工维修，具体过程如下。

（1）当巡检工到达银行把报修的设备修好后，就需要通过该银行的计算机登录该系统，对报修记录进行确认。巡检工一登录该系统就会出现如图 27.47 所示的显示该银行报修的记录。



图 27.47　报修记录

（2）单击报修记录里“操作”中“小组维修按钮”，就可以打开维修确认页面（如图 27.48 所示），在该页面中填写相应的内容后，就可以使该报修记录由“处在报修”状态转到“结束报修”状态。

（3）当巡检工对报修记录确认后，还需要银行员工登录该系统，对该记录进行确认。单击如图 27.49 所示的页面中对报修记录里“操作”中“银行确认”按钮，该报修记录就由“结束报修”状态转成“银行确认”状态。

（4）当领导想查看关于报修方面的记录时，可以通过巡检中心账户进入该系统。单击“报修管理”|“巡检中心查看报修信息”按钮就可以打开报修记录页面（如图 27.50 所示），

在该页面中单击报修记录里“操作”中的“查看”按钮，就可以打开查看该报修记录的页面（如图 27.51 所示）。

图 27.48　巡检工确认报修

图 27.49　银行确认报修

图 27.50　报修记录页面

图 27.51　查看报修记录

（5）除了银行员工可以通过该系统报修外，还可以打电话给巡检中心的值班员让其报修。当值班员想对设备报修时，可以单击该系统“报修管理”|“值班员报修”按钮，之后会弹出如图 27.52 所示的设备报修页面。在该页面中选择相应信息，则可以实现值班员的报修功能。



图 27.52　值班员报修

27.2　商业银行设备巡检系统前期准备

本节除了将详细地介绍，如何设计关于商业银行设备巡检系统的数据库和表格外，还将配置实现该系统将利用的 Struts 2.x+Spring+Hibernate+MySQL 框架的环境。其中 Struts 2.x 框架的版本号为 Struts 2.0、Spring 框架的版本号为 Spring 2.0、Hibernate 框架的版本号为 Hibernate 3.0 和数据库 MySQL 5.0。

27.2.1　设计数据库

商业银行设备巡检系统需要建立一个数据库并在该数据库中建立 14 张表，存放表格的数据库 SYYH、存放用户信息的表 users、存放岗位信息的表 Job、存放程序功能信息的表 Functions、存放系统页面信息的表 Xtymb、存放部门信息的表 Department、存放日记信息的表 Logs、存放巡检组信息的表 PI_Group、存放巡检工信息的表 PIWoker、存放银行信息的表 Bank、存放银行设备明细信息的表 Bank_Equipment、存放银行设备维修明细信息的表 EquipmentMaintain、存放银行设备类型信息的表 EquipmentType、存放银行设备报修问题类型信息的表 Fault_Repair_Type 和存放银行设备报修信息的表 Fault_Repair。

1．创建数据库SYYH

如果想创建出数据库 lams，可以在 SQL Server 2000 的查询分析器窗口中输入如下命令：

```
CREATE DATABASE 'SYYH'
```

2．创建用户表users

如果想创建出表 users（用户表），可以在 MySQL 命令窗口中输入相关命令。该表的

具体信息如表 27.1 所示。

表 27.1　表users信息

字 段 名 称	数 据 类 型	字 段 说 明
Login_ID	varchar	登录 ID
Login_Password	varchar	登录密码
User_Name	varchar	用户中文名字
Department_id	varchar	用户所在部门 ID
User_Status	char	用户状态
Job_ID	varchar	岗位代码

3．创建岗位表Job

如果想创建出表 Job（岗位表），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.2 所示。

表 27.2　表Job信息

字 段 名 称	数 据 类 型	字 段 说 明
Job_ID	varchar	岗位代码
Name	varchar	岗位中文名字
Group_ID	varchar	组代码
Description	varchar	岗位描述

4．创建程序功能表Functions

如果想创建出表 Functions（程序功能表），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.3 所示。

表 27.3　表Functions信息

字 段 名 称	数 据 类 型	字 段 说 明
Func_ID	varchar	功能代码
Func_Name	varchar	功能中文名字

5．创建系统页面表Xtymb

如果想创建出表 Xtymb（系统页面表），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.4 所示。

表 27.4　表Xtymb信息

字 段 名 称	数 据 类 型	字 段 说 明
Func_ID	varchar	功能代码
Ymbh	varchar	页面编号
Ymmc	Varchar(20)	页面控制
url	Varchar(200)	图片地址
img	Varchar(100)	图片编号

6. 创建部门表Department

如果想创建出表 Department（部门表），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.5 所示。

表 27.5　表Department信息

字 段 名 称	数 据 类 型	字 段 说 明
Department_id	varchar	部门 ID
Department_Name	varchar	部门名称

7. 创建日记表Logs

如果想创建出表 Logs（日记表），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.6 所示。

表 27.6　表Logs信息

字 段 名 称	数 据 类 型	字 段 说 明
Log_id	int	日记 ID
Checkin_time	date	登录时间
Checkout_time	date	退出时间
Users_id	VARCHAR(10)	用户 id

8. 创建巡检组表PI_Group

如果想创建出表 PI_Group（巡检组），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.7 所示。

表 27.7　表PI_Group信息

字 段 名 称	数 据 类 型	字 段 说 明
Group_id	varchar	巡检组 ID
Group_Name	varchar	巡检组名称

9. 创建巡检工表PIWoker

如果想创建出表 PIWoker（巡检工），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.8 所示。

表 27.8　表PIWoker信息

字 段 名 称	数 据 类 型	字 段 说 明
Woker_id	varchar	巡检工 ID
Worker_Name	varchar	巡检工名字
Worker_Tel_1	varchar	巡检工电话 1
Worker_Tel_2	varchar	巡检工电话 2
Group_id	varchar	巡检组 ID

10．创建银行表Bank

如果想创建出表 Bank（银行），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.9 所示。

表 27.9　表Bank信息

字 段 名 称	数 据 类 型	字 段 说 明
Bank_id	varchar	银行编码 ID
Bank_Name	varchar	银行名称
Bank_Longitude	number	银行位置经度
Bank_Latitude	number	银行位置纬度
Bank_IP	varchar	银行 IP

11．创建银行设备明细表Bank_Equipment

如果想创建出表 Bank_Equipment（银行设备明细），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.10 所示。

表 27.10　表Bank_Equipment信息

字 段 名 称	数 据 类 型	字 段 说 明
Bank_Id	varchar	银行编码 ID
Equipment_ID	varchar	设备种类 ID
EquipmentEach_ID	Varchar	设备流水 ID
Equipment_Value	Number	价值
Equipment_BuyDate	date	购入时间
Status	char	设备状态
Depreciation_Value	Number	设备折旧残值

12．创建银行设备维修明细表EquipmentMaintain

如果想创建出表 EquipmentMaintain（银行设备维修明细），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.11 所示。

表 27.11　表EquipmentMaintain信息

字 段 名 称	数 据 类 型	字 段 说 明
Equipment_ID	varchar	设备类型 ID
EquipmentEach_ID	Varchar	设备流水 ID
Maintain_Date	Date	维修日期
Maintain_Result	varchar	维修结果
字段名称	数据类型	字段说明

13．创建银行设备类型表EquipmentType

如果想创建出表 EquipmentType（银行设备类型），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.12 所示。

表 27.12　表EquipmentType信息

字 段 名 称	数 据 类 型	字 段 说 明
Equipment _id	varchar	设备种类 ID
Equipment _Name	varchar	设备名称

14．创建银行设备报修问题类型表Fault_Repair_Type

如果想创建出表 Fault_Repair_Type（银行设备报修问题类型），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.13 所示。

表 27.13　表Fault_Repair_Type信息

字 段 名 称	数 据 类 型	字 段 说 明
PITYPE _Id	varchar	设备报修问题类型 ID
PITYPE _Value	varchar	设备报修问题内容

15．创建银行设备报修表Fault_Repair

如果想创建出表 Fault_Repair（银行设备报修），可以在 MySQL 命令窗口中输入相关命令。该表的具体信息如表 27.14 所示。

表 27.14　表Fault_Repair信息

字 段 名 称	数 据 类 型	字 段 说 明
RepairID	varchar	序列号
Bank_Id	varchar	银行编码 ID
Equipment_ID	varchar	设备种类 ID
EquipmentEach_ID	Varchar	设备流水 ID
PITYPE _Id	varchar	设备报修问题类型 ID
Fault_Repair_Begin_Date	Date	设备报修时间
Fault_Repair_People_ID	varchar	设备报修人代码
Repair_Status	char	报修记录状态
Allocate_Status	char	小组分配状态
Group_ID	varchar	巡检组 ID
Fault_Repair_End_Date	Date	设备维修结束时间
Fault_Repair _Evaluation	varchar	设备维修情况描述

至此，就完成了对商业银行设备巡检系统数据库和表的设计。

27.2.2　关于 Struts 2.0 的准备

在实现 Struts　2.x 框架、Spring 框架与 Hibernate 三者集成时，对于其中的 Struts 2.0 框架除了需要引入相应的 jar 包外，还必须得对 struts.xml 和 web.xml 文件做相应的配置。

1．引入jar包

首先引入关于 Struts 2.x 框架的核心包：struts2-core-2.0.11.jar、xwork-2.0.4.jar、

ognl-2.6.11.jar、freemarker-2.3.8.jar 和 commons-logging-l.0.4.jar。然后由于该框架要与 Spring 框架整合，所以还需要 struts2-spring-plugin-2.0.8.jar。最后由于需要连接数据库 Postgresql，所以还需要引入关于该数据库的驱动 postgresql-8.3-604.jdbc4.jar。

注意：关于 Struts 2.x 框架的核心包在 Struts 2.0 框架的 jar 包中就可以找到，而最后一个关于数据库的驱动（postgresgl-8.3-604.jdbc4.jar）则必须从该数据库的官方网站下载。

2．修改web.xml文件

为了使商业银行设备巡检系统项目支持 Struts 2.0 框架，需要在 web.xml 文件中增加如代码 27.1 所示的内容。

代码 27.1　修改 web.xml 文件：web.xml

```
    <!--实现对 Spring 框架的监听-->
    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    <!--设置过滤器类-->
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <!--设置过滤器映射-->-->
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

3．创建struts.xml文件

为了使商业银行设备巡检系统项目支持 Struts 2.0 框架，需要在 votesystem/src 目录下创建 struts.xml 文件，该文件的内容如代码 27.2 所示。

代码 27.2　实现 struts.xml 文件：struts.xml

```
...
<struts>
  <!--Encoding 格式-->
  <constant name="struts.i18n.encoding" value="utf-8"></constant>
  <!--action 的扩展名-->
  <constant name="struts.action.extension" value="action,do,itfuture">
  </constant>
  <!--国际化语言版本-->
  <constant name="struts.locale" value="zh CN"></constant>
  <!--是否以开发模式来使用 struts2，如果为 true 的话，那么会在控制台输出运行信息-->
  <constant name="struts.devMode" value="true"></constant>
  <constant name="struts.objectFactory" value="spring"></constant>
  <include file="actions-xtgl.xml"></include>
  <include file="actions-sbbx.xml"></include>
```

```
    <package name="AccessSYYH" namespace="" extends="struts-default">
 ...
    </package>
</struts>
```

至此，就完成了对商业银行设备巡检系统中 Struts 2.0 框架的配置。

27.2.3　关于 Spring 2.0 的准备

在实现 Struts 2.x 框架、Spring 框架与 Hibernate 三者集成时，对于其中的 Spring 2.0 框架除了需要引入相应的 jar 包外，还必须得对 applicationContext.xml 文件做相应的配置。

1．引入jar包

首先引入关于 Spring 2.0 框架的核心包：spring.jar。

2．创建applicationContext.xml文件

首先通过 MyEclipse 开发环境中关于 Spring 框架的向导，让商业银行设备巡检系统支持 Spring 2.0 框架，然后修改 applicationContext.xml 文件支持 Spring 框架该文件的内容如代码 27.3 所示。

代码 27.3　修改 applicationContext.xml 文件：applicationContext.xml

```
...
    <!--配置 SessionFactory-->
    <bean id="sessionfactroy" class="org.springframework.orm.hibernate3.
    LocalSessionFactoryBean">
        <property name="configLocation">
            <value>classpath:hibernate.cfg.xml</value>
        </property>
    </bean>
    <bean id="transactionManager" class="org.springframework.orm.
    hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionfactroy"></property>
    </bean>
...
</beans>
```

至此，就完成了对商业银行设备巡检系统中 Spring 2.0 框架的配置。

27.2.4　关于 Hibernate 3.0 的准备

在实现 Struts 2.x 框架、Spring 框架与 Hibernate 三者集成时，对于其中的 Hibernate 3.0 框架除了需要引入相应的 jar 包外，还必须得对 hibernate.cfg.xml 文件做相应的配置。

1．引入jar包

首先引入关于 Hibernate 3.0 框架的核心包：Hibernate3.0.jar、log4j-1.2.13.jar、cglib-nodep-2.1_3.jar、dom4j-1.6.1.jar、commons-collections.jar、c3p0-0.9.0.4.jar、jta.jar、antlr-2.7.6.jar。

2．创建hibernate.cfg.xml文件

首先通过 MyEclipse 开发环境中关于 Hibernate 框架的向导，让商业银行设备巡检系统支持 Hibernate 3.0 框架，然后通过该开发环境的反向工程向导实现对数据库 SYYH 中的 14 张表进行关系数据库到对象的映射。hibernate.cfg.xml 文件的内容如代码 27.4 所示。

代码 27.4　实现 hibernate.cfg.xml 文件：hibernate.cfg.xml

```
...
<!-- Generated by MyEclipse Hibernate Tools.                 -->
<hibernate-configuration>
<session-factory>
    <!-- 指定连接数据库方言-->
    <property name="dialect">
        org.hibernate.dialect.PostgreSQLDialect
    </property>
    <!-- 指定连接数据库 URL-->
    <property name="connection.url">jdbc:postgresql://localhost:5432/
    SYYH</property>
    <!-- 指定连接数据库用户名->
    <property name="connection.username">postgres</property>
    <!-- 指定连接数据库密码->
    <property name="connection.password">123456</property>
    <!-- 指定连接数据库驱动->
    <property name="connection.driver class">
        org.postgresql.Driver
    </property>
    <property name="myeclipse.connection.profile">SYYH</property>
    <property name="format_sql">true</property>
    <property name="show sql">true</property>
    <property name="max fetch depth">1</property>
    <!-- 指定 Hibernate 映射文件 -->
    <mapping resource="com/jb/syyh/po/Bank.hbm.xml" />
    <mapping resource="com/jb/syyh/po/PiEquipmentTable.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Job.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Piwoker.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Equipmenttype.hbm.xml" />
    <mapping resource="com/jb/syyh/po/PiGroup.hbm.xml" />
    <mapping resource="com/jb/syyh/po/FaultRepair.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Department.hbm.xml" />
    <mapping resource="com/jb/syyh/po/FaultRepairType.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Gwym.hbm.xml" />
    <mapping resource="com/jb/syyh/po/BankEquipment.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Users.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Equipmentmaintain.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Functions.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Logs.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Globals.hbm.xml" />
    <mapping resource="com/jb/syyh/po/Xtymb.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

至此，就完成了对商业银行设备巡检系统中 Hibernate 3.0 框架的配置。

27.3　商业银行设备巡检系统具体实现——系统管理应用

为了让读者可以快速地理解和掌握商业银行设备巡检系统，在具体讲解时先按照面向应用的方式对该系统进行分模块：系统管理、设备巡检管理和设备报修管理，然后再对各模块个进行 MVC 分层。

本节将详细讲解系统管理应用，该模块按照 MVC 模式分为领域模型层、持久层、业务层和表现层。本节将介绍关于系统管理的相关 4 层。

27.3.1　系统管理的领域模型层

在具体实现商业银行设备巡检系统中的系统管理模块时，主要涉及的数据库表的信息如表 27.15 所示。本章将通过 MyEclipse 开发环境自动生成数据库表的领域模型对象。

表 27.15　数据库表

编号	数 据 库 表	存 储 信 息
1	表 users	存放用户信息
2	表 Job	存放岗位信息
3	表 Functions	存放程序功能信息
4	表 Xtymb	存放系统页面信息
5	表 Department	存放部门信息
6	表 Logs	存放日记信息
7	表 PI_Group	存放巡检组信息
8	表 PIWoker	存放巡检工信息
9	表 Bank	存放银行信息
10	表 Bank_Equipment	存放银行设备明细信息
11	表 EquipmentType	存放银行设备类型信息
12	表 Fault_Repair_Type	存放银行设备报修问题类型信息

1. 对应于表格users的领域模型对象

Users.java 类为表格 users 对应的领域模型对象类，该类的具体内容如代码 27.5 所示。该领域对象类的映射文件如代码 27.6 所示。

代码 27.5　表 Users 领域模型对象：Users.java

```
...
public class Users  implements java.io.Serializable {
    //创建对应字段的属性
    private String loginId;                          //登录 ID 变量
    private Department department;                   //部门变量
    private Job job;                                 //岗位变量
    private String loginPassword;                    //密码变量
    private String userName;                         //用户名变量
```

```
    private String userStatus;                          //用户状态变量
    private Set piEquipmentTables = new HashSet(0);
    private Set faultRepairs = new HashSet(0);
    public Users() {                                    //无参构造函数
    }
    //带参构造函数
    public Users(Department department, Job job, String loginPassword,
    String userName, String userStatus, Set piEquipmentTables, Set
    faultRepairs) {
       this.department = department;
       this.job = job;
       this.loginPassword = loginPassword;
       this.userName = userName;
       this.userStatus = userStatus;
       this.piEquipmentTables = piEquipmentTables;
       this.faultRepairs = faultRepairs;
}
   //省略属性 loginId、department、job、loginpassword、username、
   piequipmenttables 和 faultrepairs 的 set()和 get()方法
...
}
```

代码 27.6　Users 类的映射文件：Users.hbm.xml

```
...
<hibernate-mapping>
   <!--制定要持久化的类与对应数据库的表映射关系-->
   <class name="com.jb.syyh.po.Users" table="users" schema="public">
      <!--表 users 主键的设置-->
      <id name="loginId" type="java.lang.String">
         <column name="login_id" length="10" />
         <generator class="assigned" />
      </id>
      <!--设置多对一的映射-->
      <many-to-one name="department" class="com.jb.syyh.po.Department"
      fetch="select" lazy="false">
         <column name="department_id" />
      </many-to-one>
      <!--设置多对一的映射-->
      <many-to-one name="job" class="com.jb.syyh.po.Job" fetch="select"
      lazy="false">
         <column name="job_id" />
      </many-to-one>
      <!--表 users 字段 login_password 与属性 loginPassword 的映射关系-->
      <property name="loginPassword" type="java.lang.String">
         <column name="login_password" length="30" />
      </property>
      <!--表 users 字段 user_name 与属性 userName 的映射关系-->
      <property name="userName" type="java.lang.String">
         <column name="user_name" length="40" />
      </property>
      <!--表 users 字段 user_status 与属性 userStatus 的映射关系-->
      <property name="userStatus" type="java.lang.String">
         <column name="user_status" length="1" />
      </property>
      <set name="piEquipmentTables" inverse="true">
         <key>
            <column name="login_id" length="10" />
```

```
        </key>
        <!--设置一对多的映射-->
        <one-to-many class="com.jb.syyh.po.PiEquipmentTable" />
    </set>
    <set name="faultRepairs" inverse="true">
        <key>
            <column name="login id" length="10" />
        </key>
        <!--设置一对多的映射-->
        <one-to-many class="com.jb.syyh.po.FaultRepair" />
    </set>
  </class>
</hibernate-mapping>
```

注意：Users.java 类可以参考数据库中的表格 users 来编写，而该类的映射文件则可以通过向导实现。

2. 对应于表格Job的领域模型对象

Job.java 类为表格 Job 对应的领域模型对象类，该类的具体内容如代码 27.7 所示。该领域对象类的映射文件如代码 27.8 所示。

代码 27.7　表 Job 领域模型对象：Job.java

```
...
public class Job  implements java.io.Serializable {
    //创建对应字段的属性
    private Long jobId;                     //岗位变量
    private PiGroup piGroup;                //巡检组变量
    private String name;                    //岗位名称变量
    private String description;             //岗位描述变量
    private Set userses = new HashSet(0);
    private Set gwyms = new HashSet(0);
    public Job()                            //无参构造函数
}
    //带参构造函数
    public Job(PiGroup piGroup, String name, String description, Set userses,
    Set gwyms) {
        this.piGroup = piGroup;
        this.name = name;
        this.description = description;
        this.userses = userses;
        this.gwyms = gwyms;
}
    //省略属性 jobid、pigroup、name、description、userses 和 gwyms 的 get()和 set()
    方法
...
}
```

代码 27.8　Job 类的映射文件：Job.hbm.xml

```
...
<hibernate-mapping>
    <!--制定要持久化的类与对应数据库的表映射关系-->
    <class name="com.jb.syyh.po.Job" table="job" schema="public">
```

```
        <!--表 job 主键的设置-->
        <id name="jobId" type="java.lang.Long">
            <column name="job id" />
            <generator class="assigned" />
        </id>
        <!--设置多对一的映射-->
        <many-to-one name="piGroup" class="com.jb.syyh.po.PiGroup"
        fetch="select">
            <column name="group_id" />
        </many-to-one>
        <!--表 job 字段 name 与属性 name 的映射关系-->
        <property name="name" type="java.lang.String">
            <column name="name" length="40" />
        </property>
        <!--表 job 字段 description 与属性 description 的映射关系-->
        <property name="description" type="java.lang.String">
            <column name="description" length="100" />
        </property>
        <set name="userses" inverse="true">
            <key>
                <column name="job_id" />
            </key>
            <!--设置一对多的映射-->
            <one-to-many class="com.jb.syyh.po.Users" />
        </set>
        <set name="gwyms" inverse="true" lazy="false" cascade=
        "all-delete-orphan">
            <key>
                <column name="job id" not-null="true" />
            </key>
            <!--设置一对多的映射-->
            <one-to-many class="com.jb.syyh.po.Gwym" />
        </set>
    </class>
</hibernate-mapping>
```

注意：Job.java 类可以参考数据库中的表格 Job 来编写，而该类的映射文件则可以通过向导实现。

3. 对应于表格Functions的领域模型对象

Functions.java 类为表格 Functions 对应的领域模型对象类，该类的具体内容如代码 27.9 所示。该领域对象类的映射文件如代码 27.10 所示。

代码 27.9　表 Functions 领域模型对象：Functions.java

```
...
public class Functions  implements java.io.Serializable {
    //创建对应字段的属性
    private Long funcId;                                  //功能 ID 变量
    private String funcName;                              //功能名称变量
    private Set xtymbs = new HashSet(0);
    public Functions() {                                  //无参构造函数
}
    //有参构造函数
    public Functions(String funcName, Set xtymbs) {
```

```
        this.funcName = funcName;
        this.xtymbs = xtymbs;
}
    //省略属性 funcid、funcname 和 xtymbs 的 set()和 get()的方法
...
}
```

代码 27.10　Functions 类的映射文件：Functions.hbm.xml

```
...
<hibernate-mapping>
    <!--制定要持久化的类与对应数据库的表映射关系-->
    <class name="com.jb.syyh.po.Functions" table="functions" schema=
    "public">
        <!--表 functions 主键的设置-->
        <id name="funcId" type="java.lang.Long">
            <column name="func id" />
           <generator class="sequence" />
        </id>
        <!--表 functions 字段 func name E 与属性 funcName 的映射关系-->
        <property name="funcName" type="java.lang.String">
            <column name="func_name" length="40" />
        </property>
        <set name="xtymbs" inverse="true">
            <key>
                <column name="func id" />
            </key>
            <!--设置一对多的映射-->
            <one-to-many class="com.jb.syyh.po.Xtymb" />
        </set>
    </class>
</hibernate-mapping>
```

注意：Functions.java 类可以参考数据库中的表格 Functions 来编写，而该类的映射文件则可以通过向导实现。

4．对应于表格Xtymb的领域模型对象

Xtymb.java 类为表格 Xtymb 对应的领域模型对象类，该类的具体内容如代码 27.11 所示。该领域对象类的映射文件如代码 27.12 所示。

代码 27.11　表 Xtymb 领域模型对象：Xtymb.java

```
...
public class Xtymb  implements java.io.Serializable {
    //创建对应字段的属性
    private Long ymbh;                                  //页面编号变量
    private Functions functions;                        //功能变量
    private String ymmc;                                //页面控制变量
    private String url;                                 //页面地址变量
    private String img;                                 //页面图像变量
    private Set gwyms = new HashSet(0);
    public Xtymb() {                                    //无参构造函数
}
    //带参构造函数
```

```
    public Xtymb(Functions functions, String ymmc, String url, String img,
    Set gwyms) {
        this.functions = functions;
        this.ymmc = ymmc;
        this.url = url;
        this.img = img;
        this.gwyms = gwyms;
    }
    //省略属性 ymbh、functions、ymmc、url、img 和 gwyms 的 get()和 set()方法
...
}
```

代码 27.12　Xtymb 类的映射文件：Xtymb.hbm.xml

```
...
<hibernate-mapping>
    <!--制定要持久化的类与对应数据库的表映射关系-->
    <class name="com.jb.syyh.po.Xtymb" table="xtymb" schema="public">
        <!--表 xtymb 主键的设置-->
        <id name="ymbh" type="java.lang.Long">
            <column name="ymbh" />
            <generator class="sequence" />
        </id>
        <!--设置多对一映射关系-->
        <many-to-one name="functions" class="com.jb.syyh.po.Functions"
        fetch="select">
            <column name="func id" />
        </many-to-one>
        <!--表 xtymb 字段 ymmc 与属性 ymmc 的映射关系-->
        <property name="ymmc" type="java.lang.String">
            <column name="ymmc" length="20" />
        </property>
        <!--表 xtymb 字段 url 与属性 url 的映射关系-->
        <property name="url" type="java.lang.String">
            <column name="url" length="200" />
        </property>
        <!--表 xtymb 字段 img 与属性 img 的映射关系-->
        <property name="img" type="java.lang.String">
            <column name="img" length="100" />
        </property>
        <set name="gwyms" inverse="true">
            <key>
                <column name="ymbh" not-null="true" />
            </key>
            <!--设置一对多映射关系-->
            <one-to-many class="com.jb.syyh.po.Gwym" />
        </set>
    </class>
</hibernate-mapping>
```

注意：Xtymb.java 类可以参考数据库中的表格 Xtymb 来编写，而该类的映射文件则可以通过向导实现。

5．对应于表格Department的领域模型对象

Department.java 类为表格 Department 对应的领域模型对象类，该类的具体内容如代码

27.13 所示。该领域对象类的映射文件如代码 27.14 所示。

代码 27.13　表 Department 领域模型对象：Department.java

```
...
public class Department  implements java.io.Serializable {
   //创建对应字段的属性
   private Long departmentId;              //部门 ID 变量
   private String departmentName;          //部门名称变量
   private Set userses = new HashSet(0);
   public Department() {                   //无参构造函数
}
   //带参构造函数
   public Department(String departmentName, Set userses) {
      this.departmentName = departmentName;
      this.userses = userses;
   }
   //省略属性 departmentid、departmentname 和 userses 的 get()和 set()方法
...
}
```

代码 27.14　Department 类的映射文件：Department.hbm.xml

```
...
<hibernate-mapping>
   <!--制定要持久化的类与对应数据库的表映射关系-->
   <class name="com.jb.syyh.po.Department" table="department"
   schema="public">
      <!-- 表 Department 主键的设置-->
      <id name="departmentId" type="java.lang.Long">
         <column name="department id" />
         <generator class="sequence" />
      </id>
      <!--表 Department 字段 department name 与属性 departmentName
      的映射关系-->
      <property name="departmentName" type="java.lang.String">
         <column name="department name" length="40" />
      </property>
      <set name="userses" inverse="true">
         <key>
            <column name="department id" />
         </key>
         <!--设置一对多映射 -->
         <one-to-many class="com.jb.syyh.po.Users" />
      </set>
   </class>
</hibernate-mapping>
```

注意：Department.java 类可以参考数据库中的表格 Department 来编写，而该类的映射文件则可以通过向导实现。

6．对应于表格Logs的领域模型对象

Logs.java 类为表格 Logs 对应的领域模型对象类，该类的具体内容如代码 27.15 所示。

该领域对象类的映射文件如代码 27.16 所示。

代码 27.15　表 Logs 领域模型对象：Logs.java

```
...
public class Logs  implements java.io.Serializable {
    //创建对应字段的属性
    private Long logId;                         //日记 ID 变量
    private Date checkinTime;                   //登录时间变量
    private Date checkoutTime;                  //退出时间变量
    private String usersId;                     //用户 ID 变量
    public Logs() {                             //无参构造函数
    }
    //带参构造函数
    public Logs(Date checkinTime, Date checkoutTime, String usersId) {
        this.checkinTime = checkinTime;
        this.checkoutTime = checkoutTime;
        this.usersId = usersId;
    }
    //省略属性 logid、checkintime、checkouttime 和 usersid 的 set()和 get()方法
...
}
```

代码 27.16　Logs 类的映射文件：Logs.hbm.xml

```
...
<hibernate-mapping>
    <!--制定要持久化的类与对应数据库的表映射关系-->
    <class name="com.jb.syyh.po.Logs" table="logs" schema="public">
        <!--表 Logs 主键的设置-->
        <id name="logId" type="java.lang.Long">
            <column name="log_id" />
            <generator class="sequence" />
        </id>
        <!--表 Logs 字段 checkin_time 与属性 checkinTime 的映射关系-->
        <property name="checkinTime" type="java.util.Date">
            <column name="checkin time" length="29" />
        </property>
        <!--表 Logs 字段 checkout_time 与属性 checkinTime 的映射关系-->
        <property name="checkoutTime" type="java.util.Date">
            <column name="checkout time" length="29" />
        </property>
        <!--表 Logs 字段 users_id 与属性 usersId 的映射关系-->
        <property name="usersId" type="java.lang.String">
            <column name="users_id" length="10" />
        </property>
    </class>
</hibernate-mapping>
```

注意：Logs.java 类可以参考数据库中的表格 Logs 来编写，而该类的映射文件则可以通过向导实现。

7．对应于表格PI_Group的领域模型对象

PIGroup.java 类为表格 PI_Group 对应的领域模型对象类，该类的具体内容如代码 27.17

所示。该领域对象类的映射文件如代码 27.18 所示。

代码 27.17　表 PI_Group 领域模型对象：PIGroup.java

```
...
public class PiGroup  implements java.io.Serializable {
    //创建对应字段的属性
    private Long groupId;                          //巡检组 ID 变量
    private String groupName;                      //巡检组名称变量
    private Set faultRepairs = new HashSet(0);
    private Set piEquipmentTables = new HashSet(0);
    private Set piwokers = new HashSet(0);
    private Set jobs = new HashSet(0);
    public PiGroup() {                             //无参构造函数
    }
    //带参构造函数
    public PiGroup(String groupName, Set faultRepairs, Set piEquipmentTables,
    Set piwokers, Set jobs) {
        this.groupName = groupName;
        this.faultRepairs = faultRepairs;
        this.piEquipmentTables = piEquipmentTables;
        this.piwokers = piwokers;
        this.jobs = jobs;
    }
    //省略属性 groupid、groupname、faultrepairs、piequipmenttables、piwokershe、
    piwokers 和 jobs 的 get()和 set()方法
...
}
```

代码 27.18　PIGroup 类的映射文件：PIGroup.hbm.xml

```
...
<hibernate-mapping>
    <!--制定要持久化的类与对应数据库的表映射关系-->
    <class name="com.jb.syyh.po.PiGroup" table="pi group" schema="public">
        <!--表 PIGroup 主键的设置-->
        <id name="groupId" type="java.lang.Long">
            <column name="group id" />
            <generator class="sequence" />
        </id>
        <!--表 PIGroup 字段 group name 与属性 groupName 的映射关系-->
        <property name="groupName" type="java.lang.String">
            <column name="group_name" length="40" />
        </property>
        <set name="faultRepairs" inverse="true">
            <key>
                <column name="group id" />
            </key>
            <!-- -->
            <one-to-many class="com.jb.syyh.po.FaultRepair" />
        </set>
        <set name="piEquipmentTables" inverse="true">
            <key>
                <column name="group_id" />
            </key>
            <!--设置一对多映射-->
            <one-to-many class="com.jb.syyh.po.PiEquipmentTable" />
        </set>
```

```
        <set name="piwokers" inverse="true" cascade="all">
            <key>
                <column name="group id" />
            </key>
            <!--设置一对多映射 -->
            <one-to-many class="com.jb.syyh.po.Piwoker" />
        </set>
        <set name="jobs" inverse="true">
            <key>
                <column name="group id" />
            </key>
            <!--设置一对多映射 -->
            <one-to-many class="com.jb.syyh.po.Job" />
        </set>
    </class>
</hibernate-mapping>
```

注意：PIGroup.java 类可以参考数据库中表格 PI_Group 来编写，而该类的映射文件则可以通过向导实现。

8．对应于表格PIWoker的领域模型对象

PIWoker.java 类为表格 PIWoker 对应的领域模型对象类，该类的具体内容如代码 27.19 所示。该领域对象类的映射文件如代码 27.20 所示。

代码 27.19　表 PIWoker 领域模型对象：PIWoker.java

```
...
public class Piwoker  implements java.io.Serializable {
    //创建对应字段的属性
    private Long wokerId;                           //巡检工 ID 变量
    private PiGroup piGroup;                        //巡检组变量
    private String workerName;                      //巡检工名称变量
    private String workerTel1;                      //巡检工电话 1 变量
    private String workerTel2;                      //巡检工电话 2 变量
    public Piwoker() {                              //无参构造函数
    }
    //带参构造函数
    public Piwoker(PiGroup piGroup, String workerName, String workerTel1,
    String workerTel2) {
        this.piGroup = piGroup;
        this.workerName = workerName;
        this.workerTel1 = workerTel1;
        this.workerTel2 = workerTel2;
    }
    //省略属性 wokerld、pigroup、workername、workertel1 和 workertel2 的 set()
    和 get()方法
...
}
```

代码 27.20　PIWoker 类的映射文件：PIWoker.hbm.xml

```
...
<hibernate-mapping>
    <!--制定要持久化的类与对应数据库的表映射关系-->
```

```
    <class name="com.jb.syyh.po.Piwoker" table="piwoker" schema="public">
        <!--表 PIWoker 主键的设置-->
        <id name="wokerId" type="java.lang.Long">
            <column name="woker_id" />
           <generator class="sequence" />
        </id>
        <!--设置多对一映射关系-->
        <many-to-one name="piGroup" class="com.jb.syyh.po.PiGroup"
        fetch="select">
            <column name="group_id" />
        </many-to-one>
        <!--表 PIWoker 字段 worker_name 与属性 workerName 的映射关系-->
        <property name="workerName" type="java.lang.String">
            <column name="worker_name" length="40" />
        </property>
        <!--表 PIWoker 字段 worker_tel_1 与属性 workerTel1 的映射关系-->
        <property name="workerTel1" type="java.lang.String">
            <column name="worker_tel_1" length="40" />
        </property>
        <!--表 PIWoker 字段 worker_tel_2 与属性 workerTel2 的映射关系-->
        <property name="workerTel2" type="java.lang.String">
            <column name="worker_tel_2" length="40" />
        </property>
    </class>
</hibernate-mapping>
```

注意：PIWoker.java 类可以参考数据库中的表格 PIWoker 来编写，而该类的映射文件则可以通过向导实现。

9. 对应于表格Bank的领域模型对象

Bank.java 类为表格 Bank 对应的领域模型对象类，该类的具体内容如代码 27.21 所示。该领域对象类的映射文件如代码 27.22 所示。

代码 27.21　表 Bank 领域模型对象：Bank.java

```
...
public class Bank  implements java.io.Serializable {
    //创建对应字段的属性
    private String bankId;                          //银行 ID 变量
    private String bankName;                        //银行名称变量
    private Double bankLongitude;                   //银行精度变量
    private Double bankLatitude;                    //银行纬度变量
    private String bankIp;                          //银行 IP 变量
    private Set piEquipmentTables = new HashSet(0);
    private Set bankEquipments = new HashSet(0);
    private Set faultRepairs = new HashSet(0);
    public Bank()                                   //无参构造函数
    }
    //带参构造函数
    public Bank(String bankName, Double bankLongitude, Double bankLatitude,
    String bankIp, Set piEquipmentTables, Set bankEquipments, Set
    faultRepairs) {
        this.bankName = bankName;
        this.bankLongitude = bankLongitude;
        this.bankLatitude = bankLatitude;
```

```
        this.bankIp = bankIp;
        this.piEquipmentTables = piEquipmentTables;
        this.bankEquipments = bankEquipments;
        this.faultRepairs = faultRepairs;
    }
    //省略属性 bankid、bankname、banklongitude、banklatiitude、bankip、
    piequipmenttables、bankequipments、faultrepairs
...
}
```

代码 27.22　Bank 类的映射文件：Bank.hbm.xml

```
...
<hibernate-mapping>
    <!--制定要持久化的类与对应数据库的表映射关系-->
    <class name="com.jb.syyh.po.Bank" table="bank" schema="public">
        <!--表 Bank 主键的设置-->
        <id name="bankId" type="java.lang.String">
            <column name="bank_id" length="10" />
            <generator class="assigned" />
        </id>
        <!--表 Bank 字段 bank_name 与属性 bankname 的映射关系-->
        <property name="bankName" type="java.lang.String">
            <column name="bank name" length="40" />
        </property>
        <!--表 Bank 字段 bank longitude 与属性 banklongitude 的映射关系-->
        <property name="bankLongitude" type="java.lang.Double">
            <column name="bank longitude" precision="15" scale="10" />
        </property>
        <!--表 Bank 字段 bank_latitude 与属性 banklatitude 的映射关系-->
        <property name="bankLatitude" type="java.lang.Double">
            <column name="bank latitude" precision="15" scale="10" />
        </property>
        <!--表 Bank 字段 bank_ip 与属性 bankip 的映射关系-->
        <property name="bankIp" type="java.lang.String">
            <column name="bank ip" length="15" />
        </property>
        <set name="piEquipmentTables" inverse="true">
            <key>
                <column name="bank id" length="10" />
            </key>
            <!--设置一对多映射关系-->
            <one-to-many class="com.jb.syyh.po.PiEquipmentTable" />
        </set>
        <set name="bankEquipments" inverse="true">
            <key>
                <column name="bank id" length="10" />
            </key>
            <!--设置一对多映射关系-->
            <one-to-many class="com.jb.syyh.po.BankEquipment" />
        </set>
        <set name="faultRepairs" inverse="true">
            <key>
                <column name="bank_id" length="10" />
            </key>
            <!--设置一对多映射关系-->
```

```
        <one-to-many class="com.jb.syyh.po.FaultRepair" />
      </set>
    </class>
</hibernate-mapping>
```

注意：Bank.java 类可以参考数据库中的表格 Bank 来编写，而该类的映射文件则可以通过向导实现。

10．对应于表格Bank_Equipment的领域模型对象

BankEquipment.java 类为表格 Bank_Equipment 对应的领域模型对象类，该类的具体内容如代码 27.23 所示。该领域对象类的映射文件如代码 27.24 所示。

代码 27.23　表 Bank_Equipment 领域模型对象：BankEquipment.java

```
...
public class BankEquipment  implements java.io.Serializable {
     //创建对应字段的属性
     private String equipmenteachId;               //银行设备明细 ID 变量
     private Equipmenttype equipmenttype;          //银行设备类型变量
     private Bank bank;                            //银行变量
     private Double equipmentValue;                //银行设备价格变量
     private Date equipmentBuydate;                //银行设备购买时间变量
     private String status;                        //银行设备状态变量
     private Double depreciationValue;             //银行设备折旧变量
     private Set faultRepairs = new HashSet(0);
     private Set piEquipmentTables = new HashSet(0);
     private Set equipmentmaintains = new HashSet(0);
    public BankEquipment() {                       //无参构造函数
    }
    //带参构造函数
    public BankEquipment(Equipmenttype equipmenttype, Bank bank, Double
    equipmentValue, Date equipmentBuydate, String status, Double
    depreciationValue, Set faultRepairs, Set piEquipmentTables, Set
    equipmentmaintains) {
       this.equipmenttype = equipmenttype;
       this.bank = bank;
       this.equipmentValue = equipmentValue;
       this.equipmentBuydate = equipmentBuydate;
       this.status = status;
       this.depreciationValue = depreciationValue;
       this.faultRepairs = faultRepairs;
       this.piEquipmentTables = piEquipmentTables;
       this.equipmentmaintains = equipmentmaintains;
    }
    //配置属性 equipmenteachid、equipmenttype、bank、equipmentvalue、
    equipmentbuydate、status、depreciationvalue、faultrepairs、equipmenttype、
    Equipmentmaintains
...
     public String toString() {                    //重写 toString()方法
         return equipmenteachId+"="+equipmenteachId;
     }
}
```

代码 27.24　BankEquipment 类的映射文件：BankEquipment.hbm.xml

```
...
<hibernate-mapping>
   <!--制定要持久化的类与对应数据库的表映射关系-->
   <class name="com.jb.syyh.po.BankEquipment" table="bank equipment"
   schema="public">
      <!--表 BankEquipmen 主键的设置-->
      <id name="equipmenteachId" type="java.lang.String">
         <column name="equipmenteach id" length="10" />
         <generator class="assigned" />
      </id>
      <!--设置多对一映射关系-->
      <many-to-one name="equipmenttype" class="com.jb.syyh.po.
      Equipmenttype" lazy="false" fetch="select">
         <column name="equipment id" length="10" />
      </many-to-one>
      <!--设置多对一映射关系-->
      <many-to-one name="bank" class="com.jb.syyh.po.Bank" lazy="false"
      fetch="select">
         <column name="bank id" length="10" />
      </many-to-one>
      <!--表 BankEquipmen 字段 equipment_value 与属性 equipmentValue
      的映射关系-->
      <property name="equipmentValue" type="java.lang.Double">
         <column name="equipment value" precision="10" />
      </property>
      <!--表 BankEquipmen 字段 equipment_buydate 与属性 equipmentbuydate
      的映射关系-->
      <property name="equipmentBuydate" type="java.util.Date">
         <column name="equipment buydate" length="13" />
      </property>
      <!--表 BankEquipmen 字段 status 与属性 status 的映射关系-->
      <property name="status" type="java.lang.String">
         <column name="status" length="1" />
      </property>
      <!--表 BankEquipmen 字段 depreciation_value 与属性 depreciationvalue
      的映射关系-->
      <property name="depreciationValue" type="java.lang.Double">
         <column name="depreciation value" precision="10" />
      </property>
      <set name="faultRepairs" inverse="true" lazy="false" >
         <key>
            <column name="equipmenteach id" length="10" />
         </key>
         <!--设置一对多映射关系-->
         <one-to-many class="com.jb.syyh.po.FaultRepair" />
      </set>
      <set name="piEquipmentTables" inverse="true" lazy="false">
         <key>
            <column name="equipmenteach_id" length="10" />
         </key>
         <!--设置一对多映射关系-->
         <one-to-many class="com.jb.syyh.po.PiEquipmentTable" />
      </set>
      <set name="equipmentmaintains" inverse="true" lazy="false" >
         <key>
            <column name="equipmenteach id" length="10" />
         </key>
```

```
            <!--设置一对多映射关系-->
            <one-to-many class="com.jb.syyh.po.Equipmentmaintain" />
        </set>
    </class>
</hibernate-mapping>
```

注意：BankEquipment.java 类可以参考数据库中的表格 Bank_Equipment 来编写，而该类的映射文件则可以通过向导实现。

11．对应于表格EquipmentType的领域模型对象

EquipmentType.java 类为表格 EquipmentType 对应的领域模型对象类，该类的具体内容如代码 27.25 所示。该领域对象类的映射文件如代码 27.26 所示。

代码 27.25 表 EquipmentType 领域模型对象：EquipmentType.java

```
...
public class Equipmenttype  implements java.io.Serializable {
    //创建对应字段的属性
    private String equipmentId;                         //设备 ID 变量
    private String equipmentName;                       //设备名称变量
    private Set piEquipmentTables = new HashSet(0);
    private Set faultRepairs = new HashSet(0);
    private Set bankEquipments = new HashSet(0);
    public Equipmenttype() {                            //无参构造函数
    }
    //带参构造函数
    public Equipmenttype(String equipmentName, Set piEquipmentTables, Set
    faultRepairs, Set bankEquipments) {
        this.equipmentName = equipmentName;
        this.piEquipmentTables = piEquipmentTables;
        this.faultRepairs = faultRepairs;
        this.bankEquipments = bankEquipments;
    }
    //省略属性 equipmentid、equipmentname、piequipmenttables、faultrepairs 和
    bankequipments 的 get()和 set()方法
   ...
}
```

代码 27.26 EquipmentType 类的映射文件：EquipmentType.hbm.xml

```
...
<hibernate-mapping>
    <!--制定要持久化的类与对应数据库的表映射关系-->
    <class name="com.jb.syyh.po.Equipmenttype" table="equipmenttype"
    schema="public">
        <!--表 EquipmentType 主键的设置-->
        <id name="equipmentId" type="java.lang.String">
            <column name="equipment id" length="10" />
            <generator class="assigned" />
        </id>
        <!--表 EquipmentType 字段 equipment name 与属性 equipmentName
        的映射关系-->
        <property name="equipmentName" type="java.lang.String">
            <column name="equipment_name" length="40" />
```

```
        </property>
        <set name="piEquipmentTables" lazy="false" inverse="true">
            <key>
                <column name="equipment id" length="10" />
            </key>
            <!--设置一对多映射关系-->
            <one-to-many class="com.jb.syyh.po.PiEquipmentTable" />
        </set>
        <set name="faultRepairs" lazy="false" inverse="true">
            <key>
                <column name="equipment_id" length="10" />
            </key>
            <!--设置一对多映射关系-->
            <one-to-many class="com.jb.syyh.po.FaultRepair" />
        </set>
        <set name="bankEquipments" lazy="false" inverse="true">
            <key>
                <column name="equipment id" length="10" />
            </key>
            <!--设置一对多映射关系-->
            <one-to-many class="com.jb.syyh.po.BankEquipment" />
        </set>
    </class>
</hibernate-mapping>
```

注意： EquipmentType.java 类可以参考数据库中的表格 EquipmentType 来编写，而该类的映射文件则可以通过向导实现。

12. 对应于表格Fault_Repair_Type的领域模型对象

FaultRepairType.java 类为表格 Fault_Repair_Type 对应的领域模型对象类，该类的具体内容如代码 27.27 所示。该领域对象类的映射文件如代码 27.28 所示。

代码 27.27　表 Fault_Repair_Type 领域模型对象：FaultRepairType.java

```
...
public class FaultRepairType  implements java.io.Serializable {
    //创建对应字段的属性
    private Long pitypeId;                          //设备报修类型 ID 变量
    private String pitypeValue;                     //设备报修类型变量
    private Set piEquipmentTables = new HashSet(0);
    private Set faultRepairs = new HashSet(0);
    public FaultRepairType() {                      //无参构造函数
    }
    //带参构造函数
    public FaultRepairType(String pitypeValue, Set piEquipmentTables, Set
    faultRepairs) {
        this.pitypeValue = pitypeValue;
        this.piEquipmentTables = piEquipmentTables;
        this.faultRepairs = faultRepairs;
    }
    //省略属性 pitypeId、pitypevalue、piequipmenttables 和 faultrepairs 的 get()
    和 set()方法
...
}
```

代码 27.28　Fault-Repair-Type 类的映射文件：FaultRepairType.hbm.xml

```
...
<hibernate-mapping>
   <!--制定要持久化的类与对应数据库的表映射关系-->
   <class name="com.jb.syyh.po.FaultRepairType" table="fault repair
   type" schema="public">
      <!--表 FaultRepairType 主键的设置-->
      <id name="pitypeId" type="java.lang.Long">
         <column name="pitype id" />
         <generator class="sequence" />
      </id>
      <!--表 FaultRepairType 字段 pitype_value 与属性 pitypeValue 的映射关系-->
      <property name="pitypeValue" type="java.lang.String">
         <column name="pitype value" length="40" />
      </property>
      <set name="piEquipmentTables" lazy="false" inverse="true">
         <key>
            <column name="pitype id" />
         </key>
         <!--设置一对多映射关系-->
         <one-to-many class="com.jb.syyh.po.PiEquipmentTable" />
      </set>
      <set name="faultRepairs" lazy="false" inverse="true">
         <key>
            <column name="pitype_id" />
         </key>
         <!--设置一对多映射关系-->
         <one-to-many class="com.jb.syyh.po.FaultRepair" />
      </set>
   </class>
</hibernate-mapping>
```

注意：FaultRepairType.java 类可以参考数据库中的表格 Fault_Repair_Type 来编写，而该类的映射文件则可以通过向导实现。

至此，就完成了商业银行设备巡检系统中，系统管理模块领域模型层的设计。

27.3.2　系统管理的持久层

持久层主要用来实现对每个数据库表格进行各种操作，对于商业银行设备巡检系统中的系统管理模块的持久层，其实就是实现对各个领域模型对象的各种操作。涉及的类如表 27.16 所示。

表 27.16　持久层类

	接　口　类	实　现　类	作　　用
1	类 UsersDao	类 UsersDaoi	操作用户功能
2	类 JobDao	类 JobDaoi	操作岗位功能
3	类 FunctionsDao	类 FunctionsDaoi	操作程序功能
4	类 DepartmentDao	类 DepartmentDaoi	操作部门功能
5	类 LogsDao	类 LogsDaoi	操作日记功能
6	类 PIGroupDao	类 PIGroupDaoi	操作巡检组功能

续表

	接　口　类	实　现　类	作　　用
7	类 PIWokerDao	类 PIWokerDaoi	操作巡检工功能
8	类 BankDao	类 BankDaoi	操作银行功能
9	类 BankEquipmenDao	类 BankEquipmenDaoi	操作银行设备明细功能
10	类 EquipmentTypeDao	类 EquipmentTypeDaoi	操作银行设备类型功能
11	类 FaultRepairTypeDao	类 FaultRepairTypeDaoi	操作银行设备报修问题类型信息功能

1. 操作表格Users的相关接口和类

实现操作表 Users 的接口为 UsersDao.java，该类的具体内容如代码 27.29 所示。实现该接口的类 UsersDaoi.java，该类的具体内容如代码 27.30 所示。

代码 27.29　操作表格 Users 的接口：UsersDao.java

```
...
public interface UsersDao {
    Users findUser(String loginId,String loginPassword);//查找用户
    Users findUsersById(String loginId);        //查找用户
    List findAllUsers(int curpage,int pagerecord,String cond);
                                                //查找所有用户
    long count(String cond);                    //获取用户个数
    void updateUser(Users user);                //更新用户信息
    void mergeUser(Users user);                 //合并用户信息
    void deleteUser(String loginId);            //删除用户信息
    void save(Users user);                      //保存用户信息
    void updateUsersStatus(String loginId);     //更新用户状态
    List getUserFun(long jobId);                //获取用户相应功能
    List gerUserYms(long funcId);               //获取用户相应系统页面
    long checkUserNameExist(String loginId);    //判断用户是否退出
}
```

【代码解析】

- findUser()方法通过登录 ID 和密码实现查找用户的功能。
- findUsersById()方法通过登录 ID 来实现查找用户的功能。
- findAllUsers()方法用来查找所有用户。
- count()方法用来实现符合条件的用户个数的功能。
- updateUser()方法用来实现更新用户信息的功能。
- mergeUser()方法用来合并用户信息的功能。
- deleteUser()方法用来实现删除用户的功能。
- save()方法用来实现保存用户的功能。
- updateUsersStatus()方法用来实现更新用户状态的功能。
- getUserFun()方法用来实现获取用户的相关功能。
- gerUserYms()方法用来实现获取用户的相关系统页面。
- checkUserNameExist()方法用来验证用户是否退出。

代码 27.30　操作表格 Users 的类：UsersDaoi.java

```
...
public class UsersDaoi extends BaseDaoSupport implements UsersDao {
    public UsersDaoi() {                            //构造函数
    }
    public Users findUser(String loginId, String loginPassword) {
                                                    //通过用户名和密码获得一个用户
        String hql = "from Users A where A.loginId = '" + loginId
                + "' and A.loginPassword = '" + loginPassword + "'";
        return (Users) this.executeHQLForObject(hql);
    }
    public Users findUsersById(String loginId) {//通过用户登录 ID 获得一个用户
        return (Users) this.find(Users.class, loginId);
    }
    //实现分页展现所有的用户
    public List findAllUsers(int curpage, int pagerecord, String cond) {
        final String hql = "select new com.jb.syyh.xtgl.vo.Usersvo(A.
        loginId,A.userName,A.department.departmentName,A.job.name,
        A.userStatus) "
                + " from Users A left join A.job left join A.department "
                + cond + " order by A.loginId desc";
        return this.getPaginationDataWithHQL(hql, curpage, pagerecord);
    }
    public long count(String cond) {                //获取总记录数
        String hql = "select count(A) from Users A left join A.job left join
        A.department"+ cond;
        return this.findValue(hql);
    }
    public void deleteUser(String loginId) {        //实现删除用户功能
        this.delete(Users.class, loginId);
    }
    public void mergeUser(Users user) {             //实现合并用户属性
        this.merge(user);
    }
    public void updateUser(Users user) {            //实现更新用户信息
        this.update(user);
    }
    public void save(Users user) {                  //实现添加用户记录功能
        super.save(user);
    }
    public void updateUsersStatus(String loginId) {
        Users user = this.findUsersById(loginId);
        if ("1".equals(user.getUserStatus())) {
            user.setUserStatus("0");
        } else if ("0".equals(user.getUserStatus().trim())) {
            user.setUserStatus("1");
        } else {
            user.setUserStatus("1");
        }
        this.getHibernateTemplate().update(user);
    }
    public List getUserFun(long jobId) {
                                        //实现通过用户的岗位编号得到用户对应的模块
        String hql = "select distinct C from Gwym A, Xtymb B, Functions C
        where A.id.xtymb = B and B.functions = C and A.id.job.jobId = "
                + jobId;
        return this.executeHQL(hql);
```

```
    }
    public List gerUserYms(long funcId) {    //通过模块 ID 得到模块对应的页面
        String hql = "select distinct A from Xtymb A where A.
        functions.funcId = "
                + funcId;
        return this.executeHQL(hql);
    }
    public long checkUserNameExist(String loginId) {   //验证用户是否存在
        String hql = "select count(A) from Users A where A.loginId = '"
                + loginId + "'";
        return this.findValue(hql);
    }
}
```

注意：在上述代码中，分别实现了 UsersDao 接口中的所有方法。

2. 操作表格Job的相关接口和类

实现操作表 Job 的接口为 JobDao.java，该类的具体内容如代码 27.31 所示。实现该接口的类为 JobDaoi.java，该类的具体内容如代码 27.32 所示。

代码 27.31　操作表格 Job 的接口：JobDao.java

```
...
public interface JobDao {
    List getPageData(int curpage, int pagerecord, String cond,
            Object... objects);             //得到 Job 分页对象
    int getCount(String cond);              //得到总记录数
    Job getOneJobById(Long id);             //得到一个 Job 对象
    void saveJob(Job job);                  //保存 Job 对象
    void updateJob(Job job);                //修改 Job 对象
    void deleteOneJobById(Long id);         //通过编号删除对象
    void mergeJob(Job job);
    List getAllXtyms();                     //获取所有的页面
    List getAllFuntions();                  //获取所有的模块
    List getXtymsByFuncId(Long id);         //获取模块下的页面
    Long getMaxId();
    Long getGlId();                         //获取巡检组最大值
    List getAllJobs();                      //获取所有的岗位信息
    //根据岗位编号和模块编号获取该岗位所拥有的页面
    List getXtymsByJobIdAndFuncId(Long jobId, Long funcId);
    //通过岗位名称获取岗位信息
    Job getOneJobByJoName(String name, String nameValue);
    Xtymb getOneXtymbById(Long ymbh);       //获取新增的页面对象
    //获取一个岗位下的所有页面
    List getAllYmForJob(Long jobId, int curpage, int pagerecord);
    int getYmCount(Long id);                //获取一个岗位下所有页面的记录
}
```

【代码解析】

- ❑ getPageData()方法用来获取 Job 分页对象。
- ❑ getCount()方法用来获取总记录数。
- ❑ getOneJobById()方法用来获取一个 Job 对象。

- saveJob()方法用来保存 Job 对象。
- updateJob()方法用来修改 Job 对象。
- deleteOneJobById()方法实现通过编号删除对象。
- mergeJob()方法用来实现合并工作属性。
- getAllXtyms()方法用来获取所有的页面。
- getAllFuntions()方法用来获取所有的模块。
- getXtymsByFuncId()方法用来获取模块下的页面。
- getMaxId()方法用来获取巡检组的最大值。
- getGlId()方法用来获取所有巡检组编号。
- getAllJobs()方法用来获取所有的岗位信息。
- getXtymsByJobIdAndFuncId()方法用来实现根据岗位编号和模块编号获取该岗位所拥有的页面。
- getOneJobByJoName()方法用来实现通过岗位名称获取岗位信息。
- getOneXtymbById()方法用来获取新增的页面对象。
- getAllYmForJob()方法获取一个岗位下的所有页面。
- getYmCount()方法用来获取一个岗位下所有页面的记录。

代码 27.32　操作表格 Job 的类：JobDaoi.java

```
...
public class JobDaoi extends BaseDaoSupport implements JobDao {
    public List getAllXtyms() {                          //获取所有页面信息
        return this.findAll(Xtymb.class);
    }
    public void deleteOneJobById(Long id) {              //删除 Job 对象
        Job job = (Job)this.getHibernateTemplate().get(Job.class, id);
       this.getHibernateTemplate().delete(job);
    }
    public int getCount(String cond) {                   //获取 Job 的总记录
        int count = ((Long)this.getDataCount(Job.class)).intValue();
        return count;
    }
    public Job getOneJobById(Long id) {                  //获取一个 Job 对象
        return (Job)this.getHibernateTemplate().get(Job.class, id);
    }
    public List getPageData(int curpage, int pagerecord, String cond,
            Object... objects) {                         //获取一页数据
        String hql ="from Job po left join fetch po.gwyms left join fetch
        po.userses where 1=1 " + cond+" order by po.jobId desc";
        return this.getPaginationDataWithHQL(hql, curpage, pagerecord);
    }
    public void mergeJob(Job job) {                      //合并 Job 对象
        this.getHibernateTemplate().merge(job);
    }
    public void saveJob(Job job)                         //新增 Job
       this.getHibernateTemplate().save(job);
    }
    public void updateJob(Job job) {                     //修改 Job
         this.getHibernateTemplate().update(job);
    }
    public List getAllFuntions() {                       //获取所有模块
        return this.findAll(Functions.class);
```

```
    }
    public Long getMaxId()              //获取 Job 主键大于 2000 的最大值
            String hql = "select max(po.jobId) from Job po ";
            Long maxValue = Long.valueOf(this.executeHQLForObject
            (hql).toString());
            if(maxValue<2000){
                maxValue=2000L;
            }
            maxValue = maxValue+1;      //大于 2000 最大值加 1
            return maxValue;
        }
    public Long getGlId() {             //获取 Job 主键小于 2000 的最大值
            String hql = "select max(po.jobId) from Job po where 1=1 and
            po.jobId<'2000'";
              Object obj= this.executeHQLForObject(hql);
              if(obj==null){
                  return 1000L;
              }else{
                    Long maxValue = Long.valueOf(obj.toString());
                    maxValue = maxValue+1; //小于 2000 最大值加 1;
                     return maxValue;
              }
        }
    public List getAllJobs() {          //获取所有岗位信息
        return this.findAll(Job.class);
    }
    //根据岗位编号和模块编号获取该岗位所拥有的页面
    public List getXtymsByJobIdAndFuncId(Long jobId, Long funcId) {
          String hql="select A from Gwym A inner join fetch A.id.xtymb B inner
          join B.functions C where C.funcId=? and B = A.id.xtymb and
          A.id.job.jobId=?";
        List<Gwym> list = this.executeHQL(hql, funcId,jobId);
       return list;
    }
    //根据模块编号获取该岗位所拥有的页面
    public List getXtymsByFuncId(Long id) {
        String hql ="from Xtymb po where po.functions.id=?";
        return this.executeHQL(hql, id);
    }
    //通过岗位名称获取岗位信息
    public Job getOneJobByJoName(String name,String nameValue) {
        return (Job)this.findObject(Job.class, name, nameValue);
    }
    public Xtymb getOneXtymbById(Long ymbh) {  //通过页面编号获取页面对象信息
        return (Xtymb)this.getHibernateTemplate().get(Xtymb.class, ymbh);
    }
    //获取一个岗位下的所有页面
    public List getAllYmForJob(Long jobId,int curpage,int pagerecord) {
        String  hql="select B from Gwym A inner join A.id.xtymb B inner join
        fetch B.functions where A.id.job.jobId="+jobId +" order by B.ymbh
        desc";
        return this.find(hql, curpage, pagerecord);
    }
    public int getYmCount(Long id) {        //获取一个岗位下所有页面的记录
        String  hql="select count(B.ymbh) from Gwym A inner join A.id.xtymb
        B where A.id.job.jobId="+id;
        return this.findValue(hql).intValue();
    }
}
```

🔔注意：在上述代码中，分别实现了 JobDao 接口中的所有方法。

3. 操作表格Function的相关接口和类

实现操作表 Function 的接口为 FunctionDao.java，该类的具体内容如代码 27.33 所示。实现该接口的类为 FunctionDaoi.java，该类的具体内容如代码 27.34 所示。

代码 27.33　操作表格 Function 的接口：FunctionDao.java

```
...
public interface FunctionDao {
    //得到 Functions 分页对象
    List getPageData(int curpage, int pagerecord, String cond,
            Object... objects);
    int getCount(String cond);                          //得到总记录数
    Functions getOneFuntionById(Long id);               //得到一个 Functions 对象
    void saveFunction(Functions funtion);               //保存 Functions 对象
    void updateFunction(Functions function);            //修改 Functions 对象
    void deleteOneFunctionById(Long id);                //通过编号删除对象
    void mergeFunction(Functions function);             //合并 Functions 对象属性
    String getMaxId();                                  //获取最大值
}
```

【代码解析】

- ❑ getPageData()方法用来获取 Function 分页对象。
- ❑ getCount()方法用来获取总记录数。
- ❑ getOneFuntionById()方法实现通过 ID 后得到一个 Functions 对象。
- ❑ saveFunction()方法用来保存 Function 对象。
- ❑ updateFunction()方法用来修改 Function 对象。
- ❑ deleteOneFunctionById()方法实现通过编号删除对象。
- ❑ mergeFunction()方法用来实现合并 Function 属性。
- ❑ getMaxId()方法用来获取最大值。

代码 27.34　操作表格 Function 的类：FunctionDaoi.java

```
...
public class FunctionDaoi extends BaseDaoSupport implements FunctionDao {
    public void deleteOneFunctionById(Long id) {
                                            //通过编号删除 Functions 对象
        Functions function = (Functions) this.getHibernateTemplate().get(
                Functions.class, id);
        this.getHibernateTemplate().delete(function);
    }
    public int getCount(String cond) { //获取满足条件的记录数
        int count = ((Long) this.getDataCount(Functions.class)).
        intValue();
        return count;
    }
    public Functions getOneFuntionById(Long id) {
                                            //通过编号获取 Functions 对象
        return (Functions) this.find(Functions.class, id);
    }
```

```
    //获取一页数据
    public List getPageData(int curpage, int pagerecord, String cond,
            Object... objects) {
        return this.getPaginationData(cond, curpage, pagerecord,
                Functions.class, objects);
    }
    public void mergeFunction(Functions function) {    // merge 一个对象
        this.merge(function);
    }
    public void saveFunction(Functions function) {     //保存一个对象
        this.save(function);
    }
    public void updateFunction(Functions function) {   //修改对象
        this.update(function);
    }
    public String getMaxId() {                          //获取最大值
        String hql = "select max(po.funcId) from Functions po ";
        Long maxValue = Long.valueOf(this.getMaxId(hql).toString());
        String strValue = String.valueOf(maxValue + 1); //最大值加 1;
        return strValue;
    }
}
```

注意：在上述代码中，分别实现了 FunctionDao 接口中的所有方法。

4．操作表格Department的相关接口和类

实现操作表 Department 的接口为 DepartmentDao.java，该类的具体内容如代码 27.35 所示。实现该接口的类为 DepartmentDaoi.java，该类的具体内容如代码 27.36 所示。

代码 27.35　操作表格 Department 的接口：DepartmentDao.java

```
...
public interface DepartmentDao {
    Long count(String cond, Object... params);         //得到总记录数
    //得到一页数据
    List getPagedata(String cond, int curpage, int pagerecord, Object...
    params);
    void deleteOne(Department dept) throws Exception;  //删除某个部门
    Department getOne(Serializable id);
    void saveDept(Department dept) throws Exception;   //新增一条日志记录
    void updateDept(Department dept) throws Exception; //修改一条日志记录
    void mergeDept(Department dept) throws Exception;// merge 一条日志记录
    List getAll();
    String getId();                                     //获取 ID
    String checkDeptname(String deptname);          //验证部门名称是否存在
}
```

【代码解析】

- count()方法用来获取总记录数。
- getPagedata()方法用来获取一页数据。
- deleteOne()方法实现删除某个 dept 对象。
- getOne()方法用来获取一个 dept 对象。

- saveDept()方法用来保存 dept 对象。
- updateDept()方法用来修改 dept 对象。
- mergeDept()方法用来实现合并 dept 属性。
- getAll()方法用来获取最大值。
- getId()方法用来获取 ID 号。
- checkDeptname()方法用来验证 dept 的名字。

代码 27.36　操作表格 Department 的类：DepartmentDaoi.java

```
...
public class DepartmentDaoi extends BaseDaoSupport implements
DepartmentDao {
    public Long count(String cond, Object... params) { //获取总记录数方法
        //创建 SQL 语句
        String hql = "select count(A) from Department A where 1=1 " + cond;
        Object obj = this.getDataCountWithHQL(hql, params);//执行 SQL 语句
        return new Long(obj.toString());
    }
    public void deleteOne(Department dept) throws Exception {//删除某个 dept
        this.delete(dept);
    }
    public List getAll() {                              //获取最大值
        return this.findAll(Department.class);
    }
    public Department getOne(Serializable id) {         //获取一个 dept 对象
        return (Department) this.find(Department.class, id);
    }
    public List getPagedata(String cond, int curpage, int pagerecord,
                                                        //获取一页数据
            Object... params) {
        String hql = "select A from Department A left join fetch A.userses
        B where 1=1 "
                + cond + " order by A.departmentId desc";
        return this.getPaginationDataWithHQL(hql, curpage, pagerecord,
        params);
    }
    public void mergeDept(Department dept) throws Exception {
                                              //合并 dept 属性
        this.merge(dept);
    }
    public void saveDept(Department dept) throws Exception {
                                          //保存 dept 对象
        this.save(dept);
    }
    public void updateDept(Department dept) throws Exception {
                                          //更新 dept 对象
        this.update(dept);
    }
    public String getId() {               //获取 ID
        String hql = "select max(A.departmentId) from Department A";
                                          //创建 SQL 语句
        Object obj = this.executeHQLForObject(hql);     //执行 SQL 语句
        if (obj == null) {                              //判断 obj 对象
            return "1";
        } else {
            return new Integer(obj.toString()) + 1 + "";
```

```
        }
    }
    public List getAllDepartments() {                   //获取所有的部门
        return this.findAll(Department.class);
    }
    public String checkDeptname(String deptname) {      //校验部门名字
        String hql = "select A from Department A where A.departmentName='"
                + deptname + "'";                       //创建 SQL 语句
        Object obj = this.executeHQLForObject(hql);     //执行 SQL 语句
        if (obj == null) {                              //判断 obj 对象
            return "0";
        } else {
            return "1";
        }
    }
}
```

注意：在上述代码中，分别实现了 DepartmentDao 接口中的所有方法。

5. 操作表格Logs的相关接口和类

实现操作表 Logs 的接口为 LogsDao.java，该类的具体内容如代码 27.37 所示。实现该接口的类的 LogsDaoi.java，该类的具体内容如代码 27.38 所示。

代码 27.37　操作表格 Logs 的接口：LogsDao.java

```
...
public interface LogsDao {
    Long count(String cond, Object... params);     //得到总记录数
    //得到一页数据
    List getPagedata(String cond, int curpage, int pagerecord, Object...
    params);
    void deleteAll() throws Exception;              //全部删除日志
    void deleteOne(Logs log) throws Exception;      //删除某个日志
    Logs getOne(Long id);                           //得到唯一的一条 log 记录
    Serializable saveLogs(Logs log) throws Exception;  //新增一条日志
    void updateLogs(Logs log) throws Exception;        //修改一条日志记录
    void mergeLogs(Logs log) throws Exception;         //merge 一条日志记录
    List getAll();
}
```

【代码解析】

- count()方法用来获取总记录数。
- getPagedata()方法用来获取一页数据。
- deleteAll()方法实现删除所有 logs 对象。
- deleteOne()方法用来删除一个 logs 对象。
- getOne()方法用来得到一个 logs 对象。
- saveLogs()方法用来保存 logs 对象。
- updateLogs()方法用来更新 logs 属性。
- mergeLogs()方法用来合并 logs 属性。
- getAll()方法用来获取所有的 logs 对象。

代码 27.38　操作表格 Logs 的类：LogsDaoi.java

```
...
public class LogsDaoi extends BaseDaoSupport implements LogsDao {
    public Long count(String cond, Object... params) {//获取总记录数
        String hql = "select count(A) from Logs A where 1=1 " + cond;
                                                    //创建 SQL 语句
        Object obj = this.getDataCountWithHQL(hql, params);//执行 SQL 语句
        return new Long(obj.toString());
    }
    public void deleteAll() throws Exception {         //删除所有日记
        String hql = "delete from Logs A ";             //创建 SQL 语句
        this.executeHQLUpdate(hql);                     //执行 SQL 语句
    }
    public void deleteOne(Logs log) throws Exception {//删除一个日记记录
        this.delete(log);
    }
    public Logs getOne(Long id) {                       //获取一个日记记录
        return (Logs) this.find(Logs.class, id);
    }
    public List getPagedata(String cond, int curpage, int pagerecord,
                                                    //用来获取一页数据
            Object... params) {
        String hql = "select A from Logs A where 1=1 " + cond
                + " order by A.logId desc";             //创建 SQL 语句
        return this.getPaginationDataWithHQL(hql, curpage, pagerecord,
        params);
    }
    public void mergeLogs(Logs log) throws Exception {//合并日记属性
        this.merge(log);
    }
    public Serializable saveLogs(Logs log) throws Exception {
                                                    //添加日记方法
        return this.save(log);
    }
    public void updateLogs(Logs log) throws Exception {//更新日记方法
        this.update(log);
    }
    public List getAll() {                              //获取所有日记
        return this.findAll(Logs.class);
    }
}
```

注意：在上述代码中，分别实现了 LogsDao 接口中的所有方法。

6. 操作表格Pi_Group的相关接口和类

实现操作表 Pi_Group 的接口为 PiGroupDao.java，该类的具体内容如代码 27.39 所示。实现该接口的类为 PiGroupDaoi.java，该类的具体内容如代码 27.40 所示。

代码 27.39　操作表格 Pi_Group 的接口：PiGroupDao.java

```
...
public interface PiGroupDao {
    //巡检组分页
    List getPiGroups(int curpage, int pagerecord, String cond, Object...
```

```
params);
    int count(String cond, Object... params);       //得到巡检组总记录
    void savePiGroup(PiGroup pigroup);              //新增巡检组
    void updatePiGroup(PiGroup pigroup);            //修改巡检组
    void deletePiGroup(Long id);                    //通过编号删除巡检组
    PiGroup getOnePiGroup(Long id);                 //得到一个巡检组
    List getAllPiGroups();                          //得到所有巡检组
}
```

【代码解析】

- ❑ getPiGroups()方法实现获取巡检组分页。
- ❑ count()方法用来获取一页数据。
- ❑ savePiGroup()方法实现保存巡检组信息。
- ❑ updatePiGroup()方法用来更新巡检组信息。
- ❑ deletePiGroup()方法用来删除巡检组。
- ❑ getOnePiGroup()方法用来获取一个巡检组。
- ❑ getAllPiGroups()方法用来获取所有的巡检组。

代码 27.40　操作表格 Pi_Group 的类：PiGroupDaoi.java

```
...
public class PiGroupDaoi extends BaseDaoSupport implements PiGroupDao {
    public PiGroupDaoi() {
    }
    public int count(String cond, Object... params) {  //得到巡检组总记录
        String HQL = "select count(A) from PiGroup A where 1=1" + cond;
        int count = (new Long(this.getDataCountWithHQL(HQL,
        params).toString()))
                .intValue();
        return count;
    }
    public List getPiGroups(int curpage, int pagerecord, String cond,
                                                        //巡检组分页
            Object... params) {
        String HQL = "select A  from PiGroup A  left join fetch A.piwokers
        B where 1=1"
                + cond + " order by A.groupId desc";
        return this.getPaginationDataWithHQL(HQL, curpage, pagerecord,
        params);
    }
    public void deletePiGroup(Long id) {                //通过编号删除巡检组
        PiGroup pigroup = (PiGroup) this.find(PiGroup.class, id);
        this.delete(pigroup);
    }
    public void savePiGroup(PiGroup pigroup) {          //新增巡检组
        this.save(pigroup);
    }
    public void updatePiGroup(PiGroup pigroup) {        //修改巡检组
        this.update(pigroup);
    }
    public PiGroup getOnePiGroup(Long id) {             //得到一个巡检组
        PiGroup pigroup = (PiGroup) this.find(PiGroup.class, id);
        return pigroup;
    }
    public List getAllPiGroups() {
```

```
        String hql = "select distinct A  from PiGroup A  left join fetch
        A.piwokers B ";
        return this.executeHQL(hql);
    }
}
```

注意：在上述代码中，分别实现了 PiGroupDao 接口中的所有方法。

7. 操作表格Piwoker的相关接口和类

实现操作表 Piwoker 的接口为 PiwokerDao.java，该类的具体内容如代码 27.41 所示。实现该接口的类为 PiwokerDaoi.java，该类的具体内容如代码 27.42 所示。

代码 27.41　操作表格 Piwoker 的接口：PiwokerDao.java

```
...
public interface PiwokerDao {
    List getPiwokers(int curpage, int pagerecord, String cond);
                                               //巡检工基本信息分页
    int count(String cond);                    //得到巡检工基本信息总记录数
    void savePiwoker(Piwoker piwoker);         //新增巡检工
    void updatePiwoker(Piwoker piwoker);       //修改巡检工
    void deletePiwoker(Long id);               //删除巡检工
    PiGroup getOnePiGroup(Long id);            //得到一个巡检组
    Piwoker getOnePiwoker(Long id);            //得到一个巡检工
    public List getAllPiwokers();              //得到所有巡检工
    List getAllPiGroups();                     //得到所有巡检组
    void merger(Piwoker piwoker);              //合并巡检组属性
    List getWorkersForOneGroup(Long group_id); //得到巡检组下工作人员
    List getWorkersNotinGroup(Long group_id);
}
```

【代码解析】

- getPiwokers()方法实现获取巡检工分页。
- count()方法用来获取一页数据。
- savePiwoker()方法实现保存巡检工信息。
- updatePiwoker()方法用来更新巡检工信息。
- deletePiwoker()方法用来删除巡检工。
- getOnePiwoker()方法用来获取一个巡检工。
- getAllPiwokers()方法用来获取所有的巡检工。
- getAllPiGroups()方法用来获取巡检工属于的巡检组。
- merger()方法用来合并巡检工的属性。
- getWorkersForOneGroup()方法用来获取巡检组下的所有巡检工。
- getWorkersNotinGroup()方法用来获取没有巡检工的巡检组。

代码 27.42　操作表格 Piwoker 的类：PiwokerDaoi.java

```
...
public class PiwokerDaoi extends BaseDaoSupport implements PiwokerDao {
    public PiwokerDaoi() {                        //构造函数
    }
```

```
    public int count(String cond) {            //得到巡检工基本信息总记录数
        String hql = " select count(A) from Piwoker A where 1=1 " + cond;
        return ((Long) this.getDataCountWithHQL(hql)).intValue();
        // int count = ((Long)this.getDataCount(Piwoker.class)).
        intValue();
        // return count;
    }
    //巡检工基本信息分页
    public List getPiwokers(int curpage, int pagerecord, String cond) {
        String hql = " from Piwoker A where 1=1 " + cond
                + " order by A.wokerId desc";
        return this.getPaginationDataWithHQL(hql, curpage, pagerecord);
    }
    public void savePiwoker(Piwoker piwoker) {          //新增巡检工
        this.save(piwoker);
    }
    public void updatePiwoker(Piwoker piwoker) {        //修改巡检工
        this.update(piwoker);
    }
    public void deletePiwoker(Long id) {                //删除巡检工
        Piwoker piwoker = (Piwoker) this.find(Piwoker.class, id);
        this.delete(piwoker);
    }
    // 得到一个巡检组
    public PiGroup getOnePiGroup(Long id) {
        PiGroup pigroup = (PiGroup) this.find(PiGroup.class, id);
        return pigroup;
    }
    public Piwoker getOnePiwoker(Long id) {             //得到一个巡检工
        return (Piwoker) this.find(Piwoker.class, id);
    }
    public List getAllPiwokers() {                      //得到所有巡检工
        return this.findAll(Piwoker.class);
    }
    public List getAllPiGroups() {                      //获取所有的巡检组
        return this.findAll(PiGroup.class);
    }
    public void merger(Piwoker piwoker) {               //合并巡检组属性
        this.merge(piwoker);
    }
    public List getWorkersForOneGroup(Long group_id) {//得到某个组下的巡检工
        String hql = "select A from Piwoker A where A.piGroup.groupId='"
                + group_id + "'";
        return this.executeHQL(hql);
    }
    public List getWorkersNotinGroup(Long group_id) {
        String hql = "select A from Piwoker A where A.piGroup.groupId != '"
                + group_id + "' or A.piGroup.groupId is null";
        return this.executeHQL(hql);
    }
}
```

注意：在上述代码中，分别实现了 PiwokerDao 接口中的所有方法。

8. 操作表格Bank的相关接口和类

实现操作表 Bank 的接口为 BankDao.java，该类的具体内容如代码 27.43 所示。实现该接口的类为 BankDaoi.java，该类的具体内容如代码 27.44 所示。

代码 27.43　操作表格 Bank 的接口：BankDao.java

```
...
public interface BankDao {
    int count(String cond);                         //得到银行总记录数
    List getBanks(int curpage, int pagerecord, String cond);
                                                    //分页展现银行网点一页的数据
    void saveBank(Bank bank);                       //新增银行网点
    void updateBank(Bank bank);                     //修改银行网点
    void deleteBank(String bankId);                 //删除银行网点
    Bank getBank(String bankId);                    //通过银行 ID 得到一个银行网点对象
    Bank getOneBank(String bankIp);                 //通过银行 IP 得到一个银行网点对象
    List<Bank> getBanks();                          //得到所有的银行网点
    String checkBankId(String bankId);              //验证银行 ID 是否合法
}
```

【代码解析】

- count()方法用来获取一页数据。
- getBanks()方法用来实现银行信息的分页。
- saveBank()方法用来保存银行信息。
- updateBank()方法用来更新银行信息。
- deleteBank()方法用来删除银行信息。
- getBank()方法实现通过银行 ID 得到一个银行网点对象。
- getOneBank()方法实现通过银行 IP 得到一个银行网点对象。
- getBanks()方法用来获取所有的银行信息。
- checkBankId()方法用来验证银行的 ID 是否合法。

代码 27.44　操作表格 Bank 的类：BankDaoi.java

```
...
public class BankDaoi extends BaseDaoSupport implements BankDao {
    public BankDaoi() {                                     //构造函数
    }
    public int count(String cond) {                         //银行网点总记录数
        String hql = "select count(*) from Bank A where 1=1" + cond;
        return ((Long) this.getDataCountWithHQL(hql)).intValue();
    }
    //展现银行网点，得到一页的银行网点
    public List getBanks(int curpage, int pagerecord, String cond) {
        String hql = "select A from Bank A  left join  fetch A.bankEquipments
        where 1=1"
                + cond + " order by A.bankId desc";
        return this.getPaginationDataWithHQL(hql, curpage, pagerecord);
    }
    public void deleteBank(String bankId) {         //删除银行网点
        this.getHibernateTemplate().delete(this.getBank(bankId));
    }
```

```
    public Bank getBank(String bankId) {    //通过银行 ID 得到一个银行网点对象
        Bank bank = (Bank) this.find(Bank.class, bankId);
        // String hql="select A from Bank A where A.bankId= ? ";
        // Bank bank=(Bank)this.executeHQLForObject(hql, bankId);
        return bank;
    }
    public void saveBank(Bank bank) {       //新增银行网点
        this.save(bank);
    }
    public void updateBank(Bank bank) {     //修改银行网点
        this.update(bank);
    }
    public List<Bank> getBanks() {          //得到所有的银行网点
        String hql = " from Bank A ";
        return this.executeHQL(hql);
    }
    public Bank getOneBank(String bankIp) {//通过银行 IP 得到一个银行网点对象
        String hql = "select A from Bank A where A.bankIp= ?";
        return (Bank) this.executeHQLForObject(hql, bankIp);
    }
    public String checkBankId(String bankId) { //校验网点 ID
        String hql = "select A from Bank A where A.bankId='" + bankId + "'";
        Object obj = this.executeHQLForObject(hql);
        if (obj == null) {
            return "0";
        } else {
            return "1";
        }
    }
}
```

注意：在上述代码中，分别实现了 BankDao 接口中的所有方法。

9. 操作表格Bank_Equipment的相关接口和类

实现操作表 Bank_Equipment 的接口为 BankEquipmentDao.java，该类的具体内容如代码 27.45 所示。实现该接口的类为 BankEquipmentDaoi.java，该类的具体内容如代码 27.46 所示。

代码 27.45　操作表格 Bank_Equipment 的接口：BankEquipmentDao.java

```
...
public interface BankEquipmentDao {
    int count(String cond);                               //设备明细总记录数
    //分页展现银行设备明细，得到一页的设备明细
    List<BankEquipment> getBankEquipments(int curpage, int pagerecord,
            String cond);
    void saveBankEquipment(BankEquipment bankEquipment);//新增银行设备明细
    void deleteBankEquip(String equipmenteachId);  //删除银行设备明细
    void updateBankEquipment(BankEquipment bankEquipment);
                                                          //修改银行设备明细
    //通过设备明细 ID 得到一个银行设备明细
    BankEquipment getBankEquipment(String equipmenteachId);
    //给一个银行网点编号得到所有的明细
    List<BankEquipment> getBankEquipments(String bankId, String
    equipmentId);
```

```
    String checkBankEquId(String bankEquId);    //验证银行设备明细是否存在
}
```

【代码解析】

- count()方法用来获取一页数据。
- getBankEquipments()方法用来获取一页的设备明细。
- saveBankEquipment()方法用来保存设备明细。
- deleteBankEquip()方法用来删除设备明细信息。
- updateBankEquipment()方法用来更新设备明细信息。
- getBankEquipment()方法实现通过设备明细 ID 得到一个银行设备明细。
- getBankEquipments()方法用来获取一个银行网点编号下的所有设备明细。
- checkBankEquId()方法用来验证设备明细 ID。

代码 27.46　操作表格 Bank_Equipment 的类：BankEquipmentDaoi.java

```
...
public class BankEquipmentDaoi extends BaseDaoSupport implements
        BankEquipmentDao {
    public BankEquipmentDaoi() {                    //构造函数
    }
    //设备明细总记录数
    public int count(String cond) {
        //创建 SQL 语句
        String hql = "select count(*) from BankEquipment A where 1=1" + cond;
        return ((Long) this.getDataCountWithHQL(hql)).intValue();
    }
    // 页展现银行设备明细，得到一页的设备明细
    public List<BankEquipment> getBankEquipments(int curpage, int
    pagerecord,
            String cond) {
        //创建 SQL 语句
        String hql = "select A from BankEquipment A where 1=1 " + cond
                + " order by A.equipmenteachId desc ";
        return this.find(hql, curpage, pagerecord);    //得到一页的设备明细
    }
    //删除银行设备明细
    public void deleteBankEquip(String equipmenteachId) {
        // 创建银行设备明细对象
        BankEquipment bankEquipment = this.getBankEquipment
        (equipmenteachId);
        his.delete(bankEquipment);                  //删除银行设备明细对象
    }
    //新增银行设备明细
    public void saveBankEquipment(BankEquipment bankEquipment) {
        this.save(bankEquipment);
    }
    //修改银行设备明细
    public void updateBankEquipment(BankEquipment bankEquipment) {
        this.update(bankEquipment);
    }
    //通过设备明细 ID 得到一个银行设备明细
    public BankEquipment getBankEquipment(String equipmenteachId) {
        //创建 SQL 语句
        String hql = "select A from BankEquipment A where A.id= ?";
```

```
        return (BankEquipment) this.executeHQLForObject(hql,
        equipmenteachId);
    }
    //通过银行编号得到该银行的设备明细
    public List<BankEquipment> getBankEquipments(String bankId,
            String equipmentId) {
        //创建 SQL 语句
        String hql = "select A from BankEquipment A where A.bank.bankId='"
                + bankId + "' and  A.equipmenttype.equipmentId='" +
                equipmentId
                + "'";
        return this.executeHQL(hql);                    //执行 SQL 语句
    }
    public String getequId() {                          //维护银行 ID 的方法
        //创建 SQL 语句
        String hql = "select max(A.equipmenteachId) from BankEquipment A";
        Object obj = this.executeHQLForObject(hql);     //执行 SQL 语句
        if (obj == null) {                              //判断 obj 对象
            return "1";
        } else {
            return new Integer(obj.toString()) + 1 + "";
        }
    }
    public String checkBankEquId(String bankEquId) {
                                                //验证银行设备明细是否存在
        String hql = "select A from BankEquipment A where
        A.equipmenteachId='"
                + bankEquId + "'";              //创建 SQL 语句
        Object obj = this.executeHQLForObject(hql);     //执行 SQL 语句
        if (obj == null) {                              //判断 obj 对象
            return "0";
        } else {
            return "1";
        }
    }
}
```

注意：在上述代码中，分别实现了 BankEquipmentDao 接口中的所有方法。

10．操作表格EquipmentType的相关接口和类

实现操作表 EquipmentType 的接口为 EquipmentTypeDao.java，该类的具体内容如代码 27.47 所示。实现该接口的类为 EquipmentTypeDaoi.java，该类的具体内容如代码 27.48 所示。

代码 27.47　操作表格 EquipmentType 的接口：EquipmentTypeDao.java

```
...
public interface EquipmentTypeDao {
    //拿到某一页所有的 EquipmentType 记录（分页）
    List getEquipmentTypes(int curpage, int pagerecord, String cond);
    //拿到 EquipmentType 表总记录数
    long count(String cond);
    //新增 Equipmenttype 一条记录
    void save(Equipmenttype equipmenttype) throws Exception;
```

```
    //删除 Equipmenttype 一条记录
    void delete(Equipmenttype equipmenttype) throws Exception;
    //修改 Equipmenttype 一条记录
    void update(Equipmenttype equipmenttype) throws Exception;
    //通过一个 Equipmenttype 编号拿到一个 PO
    Equipmenttype getEquipmenttype(String equipmentId);
    //取到该 Equipmenttype 表中的最大值 ID
    List getAllEquipmentTypes();
    String checkEquipmentId(String equipmentId);//银行设备种类 ID 是否存在
    String checkEquipmentName(String equipmentName);
                                                //银行设备种类名称是否存在
}
```

【代码解析】

- getEquipmentTypes()方法用来获取一页数据。
- count()方法用来获取 Equipmenttype 对象总记录数。
- save()方法用来保存 Equipmenttype 记录。
- delete ()方法用来删除 Equipmenttype 记录。
- update()方法用来更新 Equipmenttype 记录。
- getEquipmenttype()方法实现通过一个 Equipmenttype 编号拿到一个 PO 对象。
- getAllEquipmentTypes()方法用来获取 Equipmenttype 表中的最大值 ID。
- checkEquipmentId()方法用来验证设备 ID。
- checkEquipmentName()方法用来验证设备的名称。

代码 27.48　操作表格 EquipmentType 的类：EquipmentTypeDaoi.java

```
...
public class EquipmentTypeDaoi extends BaseDaoSupport implements
        EquipmentTypeDao {
    public EquipmentTypeDaoi() {                    //构造函数
    }
    public long count(String cond) {                //获取记录数
        //创建 SQL 语句
        String hql = "select count(A) from Equipmenttype A where 1=1 " + cond;
        return (Long) this.findValue(hql);
    }
    //获取分页的一页数据
    public List getEquipmentTypes(int curpage, int pagerecord, String cond){
        String hql = "select A from Equipmenttype A left join fetch
        A.bankEquipments B where 1=1 "
                + cond + " order by A.equipmentId";
        return this.find(hql, curpage, pagerecord);
    }
    public List getAllEquipmentTypes() {            //获取所有的设备类型
        List<Equipmenttype> list = this.findAll(Equipmenttype.class);
        return list;
    }
    //删除 Equipmenttype 表中的一条记录
    public void delete(Equipmenttype equipmenttype) throws Exception {
        this.getHibernateTemplate().delete(equipmenttype);
    }
    //通过编号拿一个 Equipmenttype 表 PO
    public Equipmenttype getEquipmenttype(String equipmentId) {
        return (Equipmenttype) this.getHibernateTemplate().get(
```

```
                Equipmenttype.class, equipmentId);
    }
    //新增 Equipmenttype 表中的一条记录
    public void save(Equipmenttype equipmenttype) throws Exception {
        this.getHibernateTemplate().save(equipmenttype);
    }
    //修改 Equipmenttype 表中的一条记录
    public void update(Equipmenttype equipmenttype) throws Exception {
        this.getHibernateTemplate().update(equipmenttype);
    }
    //银行设备种类 ID 是否存在
    public String checkEquipmentId(String equipmentId) {
        String hql = "select A from Equipmenttype A where A.equipmentId='"
                + equipmentId + "'";                    //创建 SQL 语句
        Object obj = this.executeHQLForObject(hql);     //执行 SQL 语句
        if (obj == null) {                              //判断 obj 对象
            return "0";
        } else {
            return "1";
        }
    }
    public String checkEquipmentName(String equipmentName) {
        String hql = "select A from Equipmenttype A where A.equipmentName='"
                + equipmentName + "'";                  //创建 SQL 语句
        Object obj = this.executeHQLForObject(hql);     //执行 SQL 语句
        if (obj == null) {                              //判断 obj 对象
            return "0";
        } else {
            return "1";
        }
    }
}
```

注意：在上述代码中，分别实现了 EquipmentTypeDao 接口中的所有方法。

11．操作表格Fault_Repair_Type的相关接口和类

实现操作表 Fault_Repair_Type 的接口为 FaultRepairTypeDao.java，该类的具体内容如代码 27.49 所示。实现该接口的类为 FaultRepairTypeDaoi.java，该类的具体内容如代码 27.50 所示。

代码 27.49　操作表格 Fault_Repair_Type 的接口：FaultRepairTypeDao.java

```
...
public interface FaultRepairTypeDao {
    void save(FaultRepairType frt);             //新增一个设备报修问题表
    void delete(FaultRepairType frt);           //删除一个设备报修问题表
    void update(FaultRepairType frt);           //修改一条设备报修问题表
    //拿到某一页所有的 FaultRepairType 记录（分页）
    List getAllFaultRepairTypes(final int curpage, final int pagerecord,
            final String cond);
    List getAllFaultRepairTypes();              //拿到库中所有表的记录数
    List getAllFaultRepariTypes(String cond);   //模糊查询
    long count(String cond);                    //拿到设备报修表总记录数
    //通过一个 FaultRepairType 编号拿到一个 PO
```

```
    FaultRepairType getOnePo(Long pitypeId);
    String checkFaultRepairtTypeName(String pitypeValue);
                                                //校验问题名称是否存在
}
```

【代码解析】

- save()方法用来保存一个设备报修问题信息。
- delete()方法用来删除设备报修问题。
- update()方法用来更新设备报修问题。
- getAllFaultRepairTypes()方法用来获取某一页所有的设备报修问题记录。
- getAllFaultRepairTypes()方法用来获取所有的设备报修问题记录。
- getAllFaultRepariTypes()方法实现模糊查询。
- count()方法用来获取设备报修问题的记录数。
- getOnePo()方法用来获取一个设备报修信息对象。
- checkFaultRepairtTypeName()方法用来验证设备报修问题的名称。

代码 27.50　操作表格 Fault_Repair_Type 的类：FaultRepairTypeDaoi.java

```
...
public class FaultRepairTypeDaoi extends BaseDaoSupport implements
        FaultRepairTypeDao {
    public FaultRepairTypeDaoi() {                //构造函数
    }
    public void delete(FaultRepairType frt) {  //删除设备报修问题表的一条记录
        this.getHibernateTemplate().delete(frt);
    }
    public void save(FaultRepairType frt) {    //保存设备报修问题表的一条记录
        this.getHibernateTemplate().save(frt);
    }
    public void update(FaultRepairType frt) {  //修改设备报修问题表的一条记录
        this.getHibernateTemplate().update(frt);
    }
    //拿到库中的所有设备报修问题表
    public List getAllFaultRepairTypes(final int curpage,final int
    pagerecord,final String cond) {
    //创建 SQL 语句
    final String hql="from FaultRepairType A where 1=1"+cond+" order by
    A.pitypeId desc";
    return  (List)this.getHibernateTemplate().execute(new
    HibernateCallback(){
            public Object doInHibernate(Session arg0) throws
            HibernateException, SQLException {
            Query query=arg0.createQuery(hql);//执行 SQL 语句
            query.setFirstResult((curpage-1)*pagerecord);
            query.setMaxResults(pagerecord);
                return query.list();
            }
        }, false);
    }
    public long count(String cond) {        //得到 FaultRepairType 总记录数
    //创建 SQL 语句
    final String hql="select count(A) from FaultRepairType A where
    1=1"+cond;
    return (Long)this.getHibernateTemplate().execute(new
```

```
    HibernateCallback(){
       public Object doInHibernate(Session arg0) throws HibernateException,
        SQLException {
            Query query=arg0.createQuery(hql);     //执行 SQL 语句
            return query.uniqueResult();
        }
    }, false);
    }
    //通过编号拿一个设备报修问题表 PO
    public FaultRepairType getOnePo(Long pitypeId) {
        FaultRepairType frt= (FaultRepairType)this.find(FaultRepairType.
        class, pitypeId);
        return frt;
    }
    //校验问题名称是否存在
    public String checkFaultRepairtTypeName(String pitypeValue) {
                //创建 SQL 语句
                String hql ="from  FaultRepairType A where A.pitypeValue=
                '"+pitypeValue+"'";
            Object obj = this.executeHQLForObject(hql);//执行 SQL 语句
            if(obj==null){                          //判断 obj 对象
                return "0";
            }else{
                return "1";
            }
    }
    public List getAllFaultRepairTypes() {      //获取所有的请求类型
        List<FaultRepairType> list=this.findAll(FaultRepairType.class);
        return list;
    }
    public List getAllFaultRepariTypes(String cond) {
        String hql="from FaultRepairType A where 1=1 "+cond;//创建 SQL 语句
        return this.getHibernateTemplate().find(hql);
    }
}
```

注意：在上述代码中，分别实现了 FaultRepairTypeDao 接口中的所有方法。

27.3.3　系统管理的业务层

业务层主要用来实现对每个数据库表格进行各种逻辑操作，对于商业银行设备巡检系统中系统管理模块的业务层，其实就是实现该系统的各种功能。涉及的类如表 27.17 所示。

表 27.17　业务层类

编　号	接　口　类	继　承　类	作　用
1	UsersService	UsersServicei	操作用户的各种功能
2	JobService	JobServicei	实现操作岗位的各种功能
3	FunctionsService	FunctionsServicei	实现操作程序功能的各种功能
4	DepartmentService	DepartmentServicei	实现操作部门的各种功能

续表

编　　号	接　口　类	继　承　类	作　　用
5	LogsService	LogsServicei	实现操作日记的各种功能
6	PIGroupService	PIGroupServicei	实现操作巡检组的各种功能
7	PIWokerService	PIWokerServicei	实现操作巡检工的各种功能
8	BankService	BankServicei	实现操作银行的各种功能
9	BankEquipmenService	BankEquipmenServicei	实现操作银行的设备明细各种功能
10	EquipmentTypeService	EquipmentTypeServicei	实现操作银行设备类型的各种功能
11	FaultRepairTypeService	FaultRepairTypeServicei	关于设备报修问题，类型的各种操作

1. 关于users业务方面的接口和类

关于 users 操作的接口为 UsersService.java，该类的具体内容如代码 27.51 所示。实现该接口的类为 UsersServicei.java，该类的具体内容如代码 27.52 所示。

代码 27.51　实现关于 users 业务各种方法接口：UsersService.java

```
...
public interface UsersService {
    int pagerecord = 10;                   //定义每页记录数
    Users findUser(String loginId, String loginPassword);
                                           //获取用户名和密码登录
    Users findUsersById(String loginId);//通过用户 ID 得到一个具体的用户对象
    Pageinfo findAllUsers(int curpage, String cond); //满足条件的一页数据
    Bank getBank(String IP);               //根据银行编码 ID 得到具体某个银行
    void save(Users user);                 //新增用户
    void update(Users user);               //修改用户
    void merger(Users user);               //合并用户属性
    void delete(String loginId);           //删除用户
    void updateUsersState(String loginId);         //更改用户的登录状态
    List getAllJobs();                             //得到所有的岗位
    List getAllDepartments();                      //得到所有的部门
    //通过用户的岗位编号得到用户对应的模块和页面
    List getUserFun(long jobId);
    List gerUserYms(long funcId);          //通过模块 ID 得到模块对应的页面
    Serializable saveLogs(Logs log);
    Logs getOneLogs(long id);
    void updateLogs(Logs log);
    //根据岗位编号和模块编号获取该岗位所拥有的页面
    List getXtyms(long jobId, long funcId);
    long checkUserNameExist(String loginId);       //判断用户登录 ID 是否存在
}
```

【代码解析】

- findUser()方法通过用户名和密码来获取用户信息。

- findUsersById()方法通过 ID 号来获取用户信息。
- findAllUsers()方法用来获取所有的用户信息。
- getBank()方法用来获取一个银行信息。
- save()方法用来实现保存用户。
- update()方法用来实现更新用户。
- merger()方法用来合并用户属性。
- delete()方法用来实现删除用户。
- updateUsersState()方法用来更新用户状态。
- getAllJobs()方法用来获取所有的工作。
- getAllDepartments()方法用来获取所有部门。
- getUserFun()方法用来获取用户模块。
- gerUserYms()方法用来获取用户系统页面。
- saveLogs()方法用来实现保存日记信息。
- getOneLogs()方法用来实现获取一个日记信息。
- updateLogs()方法用来更新日记信息。
- getXtyms()方法用来获取所有的系统页面。
- checkUserNameExist()方法用来验证用户是否退出。

代码 27.52　实现关于 users 业务各种方法类：UsersServicei.java

```
...
public class UsersServicei implements UsersService {
    //创建各种属性
    private UsersDao usersDao;
    private BankDao bankDao;
    private JobDao jobDao;
    private DepartmentDao departmentDao;
    private LogsDao logDao;
    public UsersServicei() {                         //构造函数
    }
    public Users findUser(String loginId, String loginPassword) {
                                                     //查找用户功能
        return this.getUsersDao().findUser(loginId, loginPassword);
    }
    public Users findUsersById(String loginId) {     //通过 ID 查找用户
        return this.getUsersDao().findUsersById(loginId);
    }
    public Pageinfo findAllUsers(int curpage, String cond) {
                                                     //为分页获取所有记录
        List list = this.getUsersDao().findAllUsers(curpage, pagerecord,
        cond);
        long allrecord = this.getUsersDao().count(cond);
        Pageinfo pageinfo = new Pageinfo(curpage, (int) allrecord,
        pagerecord,list);
        return pageinfo;
    }
    public List getAllDepartments() {                //获取所有部门
        return this.getDepartmentDao().getAll();
    }
    public List getAllJobs() {                       //获取所有岗位
```

```
        return this.getJobDao().getAllJobs();
    }
    public Bank getBank(String IP) {                    //获取银行
        return this.getBankDao().getOneBank(IP);
    }
    public void delete(String loginId) {                //实现删除用户功能
        this.getUsersDao().deleteUser(loginId);
    }
    public void merger(Users user) {                    //合并用户属性
        this.getUsersDao().mergeUser(user);
    }
    public void save(Users user) {                      //添加用户记录
        this.getUsersDao().save(user);
    }
    public void update(Users user) {                    //更新用户信息
        this.getUsersDao().updateUser(user);
    }
    public void updateUsersState(String loginId) {//更新用户状态
        this.getUsersDao().updateUsersStatus(loginId);
    }
    public List getUserFun(long jobId) {                //获取功能
        return this.getUsersDao().getUserFun(jobId);
    }
    //根据岗位编号和模块编号获取该岗位所拥有的页面
    public List getXtyms(long jobId, long funcId) {
        List<Gwym> list = this.getJobDao().getXtymsByJobIdAndFuncId
        (jobId,funcId);
        List<Xtymb> xtyms = new ArrayList<Xtymb>();
        if (list != null && list.size() > 0) {
            for (Gwym gwym : list) {
                xtyms.add(gwym.getId().getXtymb());
            }
        }
        return xtyms;
    }
    public Serializable saveLogs(Logs log) {            //保存日记记录
        try {
            return this.getLogDao().saveLogs(log);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    public void updateLogs(Logs log) {                  //更新日记
        try {
            this.getLogDao().updateLogs(log);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public Logs getOneLogs(long id) {                   //获取日记
        return this.getLogDao().getOne(id);
    }
    public List gerUserYms(long funcId) {               //配置属性 useryms
        return this.getUsersDao().gerUserYms(funcId);
    }
    public long checkUserNameExist(String loginId) {
        return this.getUsersDao().checkUserNameExist(loginId);
```

```
    }
    //省略属性 usersdao、bankdao、Jobdao、logdao 和 departmentdao 的 get()
      和 set()方法
...
}
```

注意：在上述代码中，分别实现了 UsersService 接口中的所有方法。

2. 关于Job业务方面的接口和类

关于 Job 操作的接口为 JobService.java，该类的具体内容如代码 27.53 所示。实现该接口的类为 JobServicei.java，该类的具体内容如代码 27.54 所示。

代码 27.53　实现关于 Job 业务的各种方法接口：JobService.java

```
...
public interface JobService {
    int pagerecord = 10;
    //得到 Functions 分页对象
    Pageinfo getPageData(int curpage, String cond, Object... objects);
    Job getOneJobById(Long id);            //得到一个 Functions 对象
    void saveJob(Job job, String gwxz);    //保存 Functions 对象
    void updateJob(Job job);               //修改 Functions 对象
    void deleteOneJobById(Long id);        //通过编号删除对象
    List getAllXtyms();                    //得到所有的页面
    List getAllFuns();                     //得到所有的模块
    List getXtymsByFunctions(Functions function);      //获取模块下的页面
    List addYmtoGw(String[] ymbhs, Job job, Functions function);
                                           //给岗位附页面
    List getAllGroups();                   //获取所有的巡检组
    void editGroup(Job job);               //编辑巡检组
    //根据岗位编号和模块编号获取该岗位所拥有的页面
    List getXtymsByJobIdAndFuncId(Job job, Functions function);
    Job getOneJobByJoName(String name, String nameValue);
                                           //通过岗位属性的值获取 Job
    Pageinfo getAllYmForJob(Long jobId, int curpage);
                                           //获取一个岗位下的所有页面
}
```

【代码解析】

- getPageData()方法用来获取分页对象。
- getOneJobById()方法通过 ID 号获取工作信息。
- saveJob()方法用来实现保存工作信息。
- updateJob()方法用来实现更新工作信息。
- deleteOneJobById()方法用来通过 ID 号删除工作。
- getAllXtyms()方法用来获取所有的系统页面。
- getAllFuns()方法用来获取所有功能。
- getXtymsByFunctions()方法用来获取模块下的所有系统页面。
- addYmtoGw()方法用来实现给岗位附加系统页面。
- getAllGroups()方法用来获取所有的巡检组。

- editGroup()方法用来实现编辑巡检组。
- getXtymsByJobIdAndFuncId()方法用来通过根据岗位编号和模块编号，获取该岗位所拥有的系统页面。
- getOneJobByJoName()方法用来通过岗位属性的值获取 Job。
- getAllYmForJob()方法用来获取关于工作的所有系统页面。

代码 27.54　实现关于 Job 业务的各种方法类：JobServicei.java

```
...
public class JobServicei implements JobService {
    //创建属性
    private JobDao dao;
    private PiGroupDao pdao;

    public void deleteOneJobById(Long id) {         //删除 Job
        this.dao.deleteOneJobById(id);
    }
    public Job getOneJobById(Long id) {             //获取一个 Job 对象
        Job job = this.getDao().getOneJobById(id);
        Hibernate.initialize(job.getPiGroup());     //加载儿子
        return job;
    }
    //获取 Job 分页对象
    public Pageinfo getPageData(int curpage, String cond, Object... objects)
{
        int allrecord = this.getDao().getCount(cond);
        List<Job> list = this.getDao().getPageData(curpage, pagerecord,
        cond,objects);
        for (Job d : list) {
        }
        Pageinfo pageinfo = new Pageinfo(curpage, allrecord,pagerecord,
        list);
        return pageinfo;
    }
    public void saveJob(Job job, String gwxz) {    //保存 Job
        Long id = 0L;
        if ("gl".equals(gwxz)) {
            id = this.getDao().getGlId();           //获取管理岗位的编号
        } else if ("xjz".equals(gwxz)) {
            id = this.getDao().getMaxId();          //获取巡检岗位的编号
        }
        job.setJobId(id);
        this.dao.saveJob(job);
    }
    public void updateJob(Job job) {                //修改 Job
        this.getDao().updateJob(job);
    }
    //对岗位添加页面权限
    public List addYmtoGw(String[] ymbhs, Job job, Functions function) {
        //获取该 Job 对象
        Job newJob = this.getDao().getOneJobById(job.getJobId());
        List<Xtymb> newym = new ArrayList<Xtymb>();
        try {
            //获取该岗位 Job 在该模块下的页面
            List<Gwym> gwyms = this.dao.getXtymsByJobIdAndFuncId(
```

```
                    job.getJobId(), function.getFuncId());
            //删除该岗位在该模块下拥有的页面
            newJob.getGwyms().removeAll(gwyms);
            if (ymbhs != null && ymbhs.length > 0) {
                for (String ymbh : ymbhs) {
                    boolean cz = false;
                    if (gwyms != null && gwyms.size() > 0) {
                        for (Gwym gy : gwyms) {
                            if (gy.getId().getXtymb().getYmbh().equals(
                                    Long.parseLong(ymbh))) {
                                newJob.getGwyms().add(gy);
                                gwyms.remove(gy);
                                cz = true;
                                break;
                            }
                        }
                    }
                    if (cz) {
                        continue;
                    }
                    Xtymb xtym = new Xtymb();
                    xtym.setYmbh(Long.parseLong(ymbh));
                    newym.add(this.getDao().getOneXtymbById(
                            Long.parseLong(ymbh)));//获取新增页面对象
                    GwymId id = new GwymId();
                    id.setJob(job);
                    id.setXtymb(xtym);
                    Gwym gwym = new Gwym();
                    gwym.setId(id);
                    //对岗位重新添加在该模块下拥有的页面信息
                    newJob.getGwyms().add(gwym);
                }
            }
            this.dao.updateJob(newJob);
            return newym;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    public List getAllXtyms() {              //获取所有页面信息
        return this.dao.getAllXtyms();
    }
    public List getAllFuns() {               //获取所有 Functions
        return this.dao.getAllFuntions();
    }
    public List getAllGroups() {             //获取所有 Group
        return this.getPdao().getAllPiGroups();
    }
    public void editGroup(Job job) {         //编辑 Group
        Job po = this.getDao().getOneJobById(job.getJobId());
        PiGroup piGroup = job.getPiGroup();
        po.setPiGroup(piGroup);
    }
    //获取 XX 模块下所有的页面
    public List getXtymsByFunctions(Functions function) {
        return this.dao.getXtymsByFuncId(function.getFuncId());
    }
    //获取当前岗位在 XX 模块下所有的页面
    public List getXtymsByJobIdAndFuncId(Job job, Functions function) {
```

```
        List<Gwym> list = this.dao.getXtymsByJobIdAndFuncId(job.
        getJobId(),function.getFuncId());
        List ids = new ArrayList();
        if (list != null && list.size() > 0) {
            for (Gwym g : list) {
                ids.add(g.getId().getXtymb().getYmbh());
            }
        }
        return ids;
    }
    //通过岗位名称获取岗位信息
    public Job getOneJobByJoName(String name, String nameValue) {
        return this.getDao().getOneJobByJoName(name, nameValue);
    }
    public Pageinfo getAllYmForJob(Long jobId, int curpage) {
        List list = this.getDao().getAllYmForJob(jobId, curpage,
        pagerecord);
        int allrecord = this.getDao().getYmCount(jobId);
        Pageinfo pageinfo = new Pageinfo(curpage, allrecord, pagerecord,
        list);
        return pageinfo;
    }
    //省略属性 pdao 和 dao 的 set()和 get()的方法
...
}
```

注意：在上述代码中，分别实现了 JobService 接口中的所有方法。

3．关于Function业务方面的接口和类

关于 Function 操作的接口为 FunctionService.java，该类的具体内容如代码 27.55 所示。实现该接口的类为 FunctionServicei.java，该类的具体内容如代码 27.56 所示。

代码 27.55　实现关于 Function 业务的各种方法接口：FunctionService.java

```
...
public interface FunctionService {
    int pagerecord = 10;                        //定义每页的记录数
    //得到 Functions 分页对象
    Pageinfo getPageData(int curpage, String cond, Object... objects);
    Functions getOneFuntionById(Long id);   //得到一个 Functions 对象
    void saveFunction(Functions function);  //保存 Functions 对象
    void update(Functions function);        //修改 Functions 对象
    void deleteOneFunctionById(Long id);    //通过编号删除对象
}
```

【代码解析】

- getPageData()方法用来获取分页对象。
- getOneFuntionById()方法通过 ID 号来获取功能信息。
- saveFunction()方法用来实现保存功能信息。
- update()方法用来实现更新功能信息。
- deleteOneFunctionById()方法用来通过 ID 号删除功能。

代码 27.56　实现关于 Function 业务的各种方法类：FunctionServicei.java

```
...
public class FunctionServicei implements FunctionService {
    //配置属性
    private FunctionDao dao;
    public FunctionDao getDao() {       //配置属性 dao
        return dao;
    }
    public void setDao(FunctionDao dao) {
        this.dao = dao;
    }
    public void deleteOneFunctionById(Long id) {
                                   //通过编号删除 Functions 对象
        this.getDao().deleteOneFunctionById(id);
    }
    public Functions getOneFuntionById(Long id) {
                                   //通过编号获取一个 Functions 对象
        return this.dao.getOneFuntionById(id);
    }
    //Functions 分页对象
    public Pageinfo getPageData(int curpage, String cond, Object... objects){
        int allrecord = this.dao.getCount(cond);
        List list = this.dao.getPageData(curpage, pagerecord, cond,
        objects);
        Pageinfo pageinfo = new Pageinfo(curpage, allrecord, pagerecord,
        list);
        return pageinfo;
    }
    public void saveFunction(Functions function) { //保存 Functions 对象
        this.dao.saveFunction(function);
    }
    public void update(Functions function) {       //修改 Functions 对象
        this.getDao().updateFunction(function);
    }
}
```

注意：在上述代码中，分别实现了 FunctionService 接口中的所有方法。

4．关于Department业务方面的接口和类

关于 Department 操作的接口为 DepartmentService.java，该类的具体内容如代码 27.57 所示。实现该接口的类为 DepartmentServicei.java，该类的具体内容如代码 27.58 所示。

代码 27.57　实现关于 Department 业务的各种方法接口：DepartmentService.java

```
...
public interface DepartmentService {
    int pagerecord = 10;                        //定义每页的记录数
    //获取一页数据
    Pageinfo getPage(int curpage, String cond, Object... params);
    Department getOne(Serializable id);     //获取一个部门
    void saveOne(Department dept);          //保存
    void deleteOne(Department dept);        //删除
    void updateOne(Department dept);        //修改
```

```
    String checkDeptname(String deptname); //验证部门名称是否存在
}
```

【代码解析】

- getPage()方法用来获取分页对象。
- getOne()方法用来获取部门信息。
- saveOne()方法用来实现保存部门信息。
- deleteOne()方法用来实现删除部门信息。
- updateOne()方法用来更新部门信息。
- checkDeptname()方法用来验证部门。

代码 27.58　实现关于 Department 业务的各种方法类：DepartmentServicei.java

```
...
public class DepartmentServicei implements DepartmentService {
    private DepartmentDao dao;                          //创建属性 dao

    public DepartmentDao getDao() {                     //配置属性 dao
        return dao;
    }
    public void setDao(DepartmentDao dao) {
        this.dao = dao;
    }
    public void deleteOne(Department dept) {            //删除部门信息
        try {
            this.getDao().deleteOne(dept);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public Department getOne(Serializable id) {         //获取部门信息
        return (Department) this.getDao().getOne(id);
    }
    //获取分页对象
    public Pageinfo getPage(int curpage, String cond, Object... params) {
        int count = this.getDao().count(cond, params).intValue();
                                                        //获取记录数
        //获取分页对象
        List pagedata = this.getDao().getPagedata(cond, curpage, pagerecord,
                params);
        Pageinfo pageinfo = new Pageinfo(curpage, count, pagerecord,
        pagedata);
        return pageinfo;
    }
    public void saveOne(Department dept) {              //保存部门信息
        try {
            dept.setUserses(null);
            this.getDao().saveDept(dept);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void updateOne(Department dept) {            //更新部门信息
        try {
            this.getDao().updateDept(dept);
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public String checkDeptname(String deptname) {//校验部门名字信息
        return this.getDao().checkDeptname(deptname);
    }
}
```

注意：在上述代码中，分别实现了 DepartmentService 接口中的所有方法。

5. 关于Logs业务方面的接口和类

关于 Logs 操作的接口为 LogsService.java，该类的具体内容如代码 27.59 所示。实现该接口的类为 LogsServicei.java，该类的具体内容如代码 27.60 所示。

代码 27.59　实现关于 Logs 业务的各种方法接口：LogsService.java

```
...
public interface LogsService {
    int pagerecord = 10;                              //设置每页记录数
    //得到一页数据
    Pageinfo getPage(int curpage, String cond, Object... params);
    void deleteAll();                                 //删除全部的日志
    void deleteOne(Long id);                          //删除一条日志
    //创建 Excel 文件
    void createExcelForAll(List<Logs> list, HttpServletRequest request,
            HttpServletResponse response);
    void saveLogs(Logs log);                          //保存 log
    List getAll();
}
```

【代码解析】

- getPage()方法用来获取分页对象。
- deleteAll()方法用来删除所有的日记信息。
- deleteOne()方法用来实现删除一个日记信息。
- createExcelForAll()方法用来实现创建 Excel 功能。
- saveLogs()方法用来实现保存日记信息。
- getAll()方法用来获取所有日记信息。

代码 27.60　实现关于 Logs 业务的各种方法接口：LogsServicei.java

```
...
public class LogsServicei implements LogsService {
    private LogsDao dao;                          //创建属性
    public LogsDao getDao() {                     //配置属性 dao
        return dao;
    }
    public void setDao(LogsDao dao) {
        this.dao = dao;
    }
    //获取分页对象
```

```
public Pageinfo getPage(int curpage, String cond, Object... params) {
    //获取记录数
    int count = this.getDao().count(cond, params).intValue();
    List pagedata = this.getDao().getPagedata(cond, curpage, pagerecord,
            params);
    //创建 Pageinfo 对象
    Pageinfo pageinfo = new Pageinfo(curpage, count, pagerecord,
    pagedata);
    return pageinfo;
}
public void deleteAll() {                    //删除所有日记记录
    try {
        this.getDao().deleteAll();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public void deleteOne(Long id) {             //删除一条日记记录
    Logs log = this.getDao().getOne(id);     //获取符合条件的日记记录
    try {
        this.getDao().deleteOne(log);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
//为所有的记录创建 Excel 表格
public void createExcelForAll(List<Logs> list, HttpServletRequest
request,
        HttpServletResponse response) {
    this.forExcel(list, request, response);
}
public void saveLogs(Logs log) {             //添加日记记录
    try {
        this.getDao().saveLogs(log);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public List getAll() {                       //获取所有的日记记录
    return this.getDao().getAll();
}
//为生成 Excel 服务
private void forExcel(List<Logs> list, HttpServletRequest request,
        HttpServletResponse response) {
    String[][] logsStr = new String[list.size() + 1][];
    logsStr[0] = new String[] { "登录 ID", "登录时间", "退出时间" };
    int i = 1;
    String out = "";
    for (Logs log : list) {
        if (log.getCheckoutTime() != null) {
            out = DateFormate.formateData(log.getCheckoutTime(),
                    "yyyy-MM-dd HH:MM:ss");
        }
        logsStr[i++] = new String[] {
                log.getUsersId(),
                DateFormate.formateData(log.getCheckinTime(),
                        "yyyy-MM-dd HH:MM:ss"), out };
    }
    //创建工作簿
```

```
        HSSFSheet sheet = null;
        try {
            HSSFWorkbook book = new HSSFWorkbook();
            sheet = book.createSheet();
            // 设置中文编码
            book.setSheetName(0, "日志表", HSSFWorkbook.ENCODING_UTF_16);
            // 创建字体，设置其为红色、粗体
            HSSFFont font = book.createFont();
            font.setColor(HSSFFont.COLOR_RED);
            font.setBoldweight(HSSFFont.BOLDWEIGHT_BOLD);
            // 创建格式
            HSSFCellStyle cellStyle = book.createCellStyle();
            cellStyle.setFont(font);
            //在索引 0 的位置创建行（最顶端的行）
            HSSFRow row = sheet.createRow((short) 0);
            //在上面行索引 0 的位置创建单元格（左上端）
            HSSFCell cell = row.createCell((short) 0);
            //定义单元格为字符串类型
            cell.setCellType(HSSFCell.CELL_TYPE_STRING);
            //正常显示中文，需要为单元格设置编码
            cell.setEncoding((short) 1);//cell.setEncoding(HSSFCell.ENC-
            ODING_UTF_16);
            //在单元格中输入一些内容
            cell.setCellValue("登录 ID");
            //应用格式
            cell.setCellStyle(cellStyle);
            //在上面行列索引 1 的位置创建单元格（第二列）
            cell = row.createCell((short) 1);
            //定义单元格为字符串类型
            cell.setCellType(HSSFCell.CELL_TYPE_STRING);
            //正常显示中文，需要为单元格设置编码
            cell.setEncoding(HSSFCell.ENCODING_UTF_16);
            //在单元格中输入一些内容
            cell.setCellValue("登录时间");
            //应用格式
            cell.setCellStyle(cellStyle);
            cell = row.createCell((short) 2);
            //定义单元格为字符串类型
            cell.setCellType(HSSFCell.CELL_TYPE_STRING);
            //正常显示中文，需要为单元格设置编码
            cell.setEncoding(HSSFCell.ENCODING_UTF_16);
            //在单元格中输入一些内容
            cell.setCellValue("退出时间");
            //应用格式
            cell.setCellStyle(cellStyle);
            //从第二行开始是数据
            int rownum = 1;
            for (Logs info : list) {
                HSSFRow rowdata = sheet.createRow((short) rownum);
                //在上面行索引 0 的位置创建单元格
                HSSFCell celldata = rowdata.createCell((short) 0);
                //定义单元格为字符串类型
                celldata.setCellType(HSSFCell.CELL_TYPE_STRING);
                //正常显示中文，需要为单元格设置编码
                celldata.setEncoding(HSSFCell.ENCODING_UTF_16);
                //在单元格中输入一些内容
                celldata.setCellValue(info.getUsersId() + "");
```

```
                //在上面行索引 1 的位置创建单元格
                celldata = rowdata.createCell((short) 1);
                //定义单元格为字符串类型
                celldata.setCellType(HSSFCell.CELL TYPE STRING);
                //正常显示中文，需要为单元格设置编码
                celldata.setEncoding(HSSFCell.ENCODING_UTF_16);
                //在单元格中输入一些内容
                celldata.setCellValue(DateFormate.formateData(info
                        .getCheckinTime(), "yyyy-MM-dd HH:MM:ss"));
                //在上面行索引 2 的位置创建单元格
                celldata = rowdata.createCell((short) 2);
                //定义单元格为字符串类型
                celldata.setCellType(HSSFCell.CELL_TYPE_STRING);
                //正常显示中文，需要为单元格设置编码
                celldata.setEncoding(HSSFCell.ENCODING UTF 16);
                if (info.getCheckoutTime() != null) {
                                                    //在单元格中输入一些内容
                    celldata.setCellValue(DateFormate.formateData(info
                            .getCheckoutTime(), "yyyy-MM-dd HH:MM:ss"));
                } else {
                    celldata.setCellValue("");
                }
                rownum++;
            }
            //在服务器端创建文件夹
            String path = request.getSession().getServletContext().
            getRealPath(
                    "/files");
            File folder = new File(path);
            if (!folder.exists()) {
                folder.mkdir();
            }
            String filepath = path + "/student.xls";   //指定文件路径和名
            //新建文件输出流到文件
            FileOutputStream fOut = new FileOutputStream(filepath);
            //把相应的 Excel 工作簿存盘(存到服务器端)
            book.write(fOut);
            fOut.flush();
            fOut.close();                                   //操作结束，关闭文件
            response.setContentType("application/vnd.ms-excel;charset=
            utf-8");
            //表示以附件的形式把文件发送到客户端
            response.setHeader("Content-Disposition", "attachment;
            filename="+ new String("student.xls".getBytes(),"ISO8859-1" ));
            //通过 response 的输出流把工作簿的流发送到浏览器形成文件
            OutputStream os = response.getOutputStream();
            book.write(os);
            os.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

注意：在上述代码中，分别实现了 LogsService 接口中的所有方法。

6．关于PiGroupService业务方面的接口和类

关于 PiGroup 操作的接口为 PiGroupService.java，该类的具体内容如代码 27.61 所示。实现该接口的类为 PiGroupServicei.java，该类的具体内容如代码 27.62 所示。

代码 27.61　实现关于 PiGroup 业务的各种方法接口：PiGroupService.java

```
...
public interface PiGroupService {
    int pagerecord = 10;                              //设置分页单位
    //得到巡检组分页信息
    Pageinfo getPiGroups(int curpage, String cond, Object... params);
    void savePiGroup(PiGroup pigroup, Long[] workids);//新增巡检组
    void updatePiGroup(PiGroup pigroup);             //修改巡检组
    void deletePiGroup(Long id);                     //删除巡检组
    PiGroup getOnePiGroup(Long id);                  //得到一个巡检组
    List getAllPiGroups();                           //得到所有巡检组
    List getAllWorks();
    List getWorks(Long id);                          //根据组的 ID 得到这个组下的人
    List getWorksNotinGroup(Long id);
}
```

【代码解析】

- ❑ getPiGroups()方法用来获取分页对象。
- ❑ savePiGroup()方法用来实现添加巡检组。
- ❑ updatePiGroup()方法用来实现更新巡检组信息。
- ❑ deletePiGroup()方法用来实现删除巡检组信息。
- ❑ getOnePiGroup()方法用来实现获取巡检组信息。
- ❑ getAllPiGroups()方法用来获取所有巡检组信息。
- ❑ getAllWorks()方法用来获取巡检组下的巡检工。
- ❑ getWorks()方法用来获取所有的巡检工。
- ❑ getWorksNotinGroup()方法用来获取不属于巡检组的巡检工。

代码 27.62　实现关于 PiGroup 业务的各种方法类：PiGroupServicei.java

```
...
public class PiGroupServicei implements PiGroupService {
    //创建属性
    private PiGroupDao dao;
    private PiwokerDao workerDao;
    public PiGroupServicei() {                             //构造函数
    }
    // 巡检组分页
    public Pageinfo getPiGroups(int curpage, String cond, Object... params)
    {
        List list = this.getDao()
                .getPiGroups(curpage, pagerecord, cond, params);
        int count = this.getDao().count(cond, params);
        Pageinfo pageinfo = new Pageinfo(curpage, count, pagerecord, list);
        return pageinfo;
```

```
    }
    public void deletePiGroup(Long id) {                //删除巡检组
        this.getDao().deletePiGroup(id);
    }
    //新增巡检组
    public void savePiGroup(PiGroup pigroup, Long[] workids) {
        if (workids != null && workids.length > 0) {   //判断 workids 对象
            Set set = new HashSet();                    //创建 Set 对象
            for (Long workid : workids) {               //遍历 workid 对象
                Piwoker woker = this.getWorkerDao().getOnePiwoker(workid);
                woker.setPiGroup(pigroup);
                set.add(woker);
            }
            pigroup.setPiwokers(set);
        }
        this.getDao().savePiGroup(pigroup);
    }
    public void updatePiGroup(PiGroup pigroup) {        //修改巡检组
        this.getDao().updatePiGroup(pigroup);
    }
    public PiGroup getOnePiGroup(Long id) {             //得到一个巡检组
        PiGroup pigroup = this.getDao().getOnePiGroup(id);
        return pigroup;
    }
    public List getAllPiGroups() {                      //得到所有巡检组
        return this.getDao().getAllPiGroups();
    }
    public List getAllWorks() {                         //获取所有的巡检组
        return this.getWorkerDao().getAllPiwokers();
    }
    public List getWorksNotinGroup(Long id) {       //获取没有巡检工的巡检组
        return this.getWorkerDao().getWorkersNotinGroup(id);
    }
    public List getWorks(Long id) {                 //获取符合条件的巡检组
        return this.getWorkerDao().getWorkersForOneGroup(id);
    }
    //省略属性 pigroupdao 和 workerdao 的 get()和 set()方法
...
}
```

注意：在上述代码中，分别实现了 PiGroupService 接口中的所有方法。

7．关于Piwoker业务方面的接口和类

关于 Piwoker 操作的接口为 PiwokerService.java，该类的具体内容如代码 27.63 所示。实现该接口的类为 PiwokerServicei.java，该类的具体内容如代码 27.64 所示。

代码 27.63　实现关于 Piwoker 业务的各种方法接口：PiwokerService.java

```
...
public interface PiwokerService {
    int pagerecord = 10;    `                       //分页单位
    Pageinfo getPiwokers(int curpage, String cond);
```

```
                                                  //得到巡检工基本信息分页
    void savePiwoker(Piwoker piwoker);            //新增巡检工
    void updatePiwoker(Piwoker piwoker);          //修改巡检工
    void deletePiwoker(Long id);                  //删除巡检工
    PiGroup getOnePiGroup(Long id);               //得到一个巡检组
    Piwoker getOnePiwoker(Long id);               //得到一个巡检工
    List getAllPiwokers();                        //得到所有巡检工
    List<Piwoker> getAllPiGroups();               //得到所有巡检组
    void updateGroupWorker(Long group id, String[] work ids);
    String forMenu(Long group_id);                //一个组下的巡检工
}
```

【代码解析】

- getPiwokers()方法用来获取分页对象。
- savePiwoker()方法用来实现添加一个巡检工。
- updatePiwoker()方法用来删除巡检工。
- deletePiwoker()方法用来删除巡检组。
- getOnePiGroup()方法用来获取一个巡检组。
- getOnePiwoker()方法用来获取一个巡检工。
- getAllPiwokers()方法用来获取所有的巡检工。
- getAllPiGroups()方法用来获取所有的巡检组。
- updateGroupWorker()方法用来更新巡检工信息。
- forMenu()方法用来获取一个组下的巡检工。

代码 27.64　实现关于 Piwoker 业务的各种方法类：PiwokerServicei.java

```
...
public class PiwokerServicei implements PiwokerService {
    //创建属性
    private PiwokerDao dao;
    private PiGroupDao groupDao;
    public PiGroupDao getGroupDao() {             //配置属性 groupDao
        return groupDao;
    }
    public void setGroupDao(PiGroupDao groupDao) {
        this.groupDao = groupDao;
    }
    public PiwokerServicei() {                    //构造函数
    }
    //分页展现巡检工基本信息
    public Pageinfo getPiwokers(int curpage, String cond) {
        List list = this.getDao().getPiwokers(curpage, pagerecord, cond);
        int count = this.getDao().count(cond);
        Pageinfo pageinfo = new Pageinfo(curpage, count, pagerecord, list);
        return pageinfo;
    }
    public PiGroup getOnePiGroup(Long id) {       //得到一个巡检组
        PiGroup pigroup = this.getDao().getOnePiGroup(id);
        return pigroup;
    }
    public void savePiwoker(Piwoker piwoker) {    //新增巡检工
        this.getDao().savePiwoker(piwoker);
    }
```

```
    public void updatePiwoker(Piwoker piwoker) {        //修改巡检工
        Piwoker po = this.getDao().getOnePiwoker(piwoker.getWokerId());
        piwoker.setPiGroup(po.getPiGroup());
        this.getDao().merger(piwoker);
    }
    public void deletePiwoker(Long id) {                //删除巡检工
        this.getDao().deletePiwoker(id);
    }
    public Piwoker getOnePiwoker(Long id) {             //得到一个巡检工
        return this.getDao().getOnePiwoker(id);
    }
    public List getAllPiwokers() {                      //得到所有巡检工
        return this.getDao().getAllPiwokers();
    }
    public List getAllPiGroups() {                      //得到所有巡检组
        return this.getDao().getAllPiGroups();
    }
    public PiwokerDao getDao() {                        //配置属性 dao
        return dao;
    }
    public void setDao(PiwokerDao dao) {
        this.dao = dao;
    }
    public void updateGroupWorker(Long group_id, String[] work_ids) {
                                                        //更新巡检工
        PiGroup group = this.getDao().getOnePiGroup(group_id);
                                                        //创建巡检组
        Set<Piwoker> works = group.getPiwokers();       //获取巡检工
        for (Piwoker woker : works) {                   //遍历巡检工
            woker.setPiGroup(null);
            this.getDao().updatePiwoker(woker);
        }
        if (work_ids != null && work_ids.length > 0) { //判断巡检工
            for (String work_id : work_ids) {
                Long id = new Long(work_id);
                Piwoker work = this.getDao().getOnePiwoker(id);
                work.setPiGroup(group);
                this.getDao().updatePiwoker(work);
            }
        }
        this.getGroupDao().updatePiGroup(group);
    }
    public String forMenu(Long group_id) {              //某个组下的巡检工
        //获取巡检工
        List<Piwoker> workers = this.getDao().getWorkersForOneGroup
        (group_id);
        StringBuffer sb = new StringBuffer();       //创建 StringBuffer 对象
        for (Piwoker woker : workers) {             //遍历巡检工
            sb.append(woker.getWokerId() + "," + woker.getWorkerName() +
            ";");
        }
        if (sb.lastIndexOf(";") != -1) {
            return sb.substring(0, sb.lastIndexOf(";"));
        } else {
            return "no";
        }
    }
}
```

注意：在上述代码中，分别实现了 PiwokerService 接口中的所有方法。

8．关于Bank业务方面的接口和类

关于 Bank 操作的接口为 BankService.java，该类的具体内容如代码 27.65 所示。实现该接口的类为 BankServicei.java，该类的具体内容如代码 27.66 所示。

代码 27.65　实现关于 Bank 业务的各种方法接口：BankService.java

```
...
public interface BankService {
    int pagerecord = 10;                             //银行网点分页单位
    Pageinfo getBanks(int curpage, String cond); //页展现所有的银行网点
    // 通过编号删除一个银行网点信息（有设备信息的银行网点不能删除）
    void deleteBank(String bankId);
    void saveBank(Bank bank);                        //新增一个银行网点
    void updateBank(Bank bank);                      //修改一个银行网点
    Bank getBank(String bankId);                     //通过银行编号得到一个银行网点
    Bank getOneBank(String bankIp);                  //通过银行 IP 得到一个银行网点
    List<Bank> getBanks();                           //得到所有的银行网点
    String checkBankId(String bankId);               //验证银行 ID 是否存在
}
```

【代码解析】

- ❑ getBanks()方法用来获取分页对象。
- ❑ deleteBank()方法用来实现删除银行信息。
- ❑ saveBank()方法用来实现保存银行信息。
- ❑ updateBank()方法用来实现更新银行信息。
- ❑ getBank()方法用来获取银行信息。
- ❑ getOneBank()方法用来获取一个银行信息。
- ❑ getBanks()方法用来获取所有的银行信息。
- ❑ checkBankId()方法用来验证银行 ID 号。

代码 27.66　实现关于 Bank 业务的各种方法类：BankServicei.java

```
...
public class BankServicei implements BankService {
    private BankDao bankDao;                           //创建属性
    public BankServicei() {                            //构造函数
    }
    public Pageinfo getBanks(int curpage, String cond) {
                                                       //分页展现所有的银行网点
        int allrecord = this.getBankDao().count(cond);
        List list = this.getBankDao().getBanks(curpage, pagerecord, cond);
        Pageinfo pageinfo = new Pageinfo(curpage, allrecord, pagerecord,
        list);
        return pageinfo;
    }
    public void deleteBank(String bankId) {            //删除一个银行网点
        this.getBankDao().deleteBank(bankId);
    }
    public void saveBank(Bank bank) {                  //新增一个银行网点
```

```
        this.getBankDao().saveBank(bank);
    }
    public void updateBank(Bank bank) {          //修改银行网点
        this.getBankDao().updateBank(bank);
    }
    public Bank getBank(String bankId) {         //通过银行编号得到银行网点
        Bank bank = this.getBankDao().getBank(bankId);
        return bank;
    }
    public List<Bank> getBanks() {               //得到所有的银行网点
        List banks = this.getBankDao().getBanks();
        return banks;
    }
    public Bank getOneBank(String bankIp) {      //通过银行 IP 得到一个银行网点
        Bank bank = this.getBankDao().getOneBank(bankIp);
        return bank;
    }
    public String checkBankId(String bankId) { //验证银行 ID 是否存在
        return this.bankDao.checkBankId(bankId);
    }

    public BankDao getBankDao() {                //配置属性 bankdao
        return bankDao;
    }
    public void setBankDao(BankDao bankDao) {
        this.bankDao = bankDao;
    }
}
```

注意：在上述代码中，分别实现了 BankService 接口中的所有方法。

9．关于BankEquipment业务方面的接口和类

关于 BankEquipment 操作的接口为 BankEquipmentService.java，该类的具体内容如代码 27.67 所示。实现该接口的类为 BankEquipmentServicei.java，该类的具体内容如代码 27.68 所示。

代码 27.67　实现关于 BankEquipment 业务的各种方法接口：BankEquipmentService.java

```
...
public interface BankEquipmentService {
    int pagerecord = 10;                                   //分页单位
    Pageinfo getBankEquipments(int curpage, String cond);
                                                      //页展现所有银行设备明细
    // 通过设备流水 ID 删除一个设备明细
    void deleteBankEquipment(String equipmenteachId);
    void saveBankEquipment(BankEquipment bankEquipment);
                                                      //新增一个银行设备明细
    void updateBankEquipment(BankEquipment bankEquipment);
                                                      //修改银行设备明细
    //通过设备流水 ID 得到一个设备明细
    BankEquipment getBankEquipment(String equipmenteachId);
    Bank getBank(String bankId);                      //通过银行编号得到银行
    // 得到所有的设备种类为新增设备明细做准备
    List getAllEquipmentTypes();
```

```
    // 传一个银行网点 ID 得到银行下的所有设备明细
    List<BankEquipment> getBankEquipments(String bankId, String
    equipmentId);
    String checkBankEquId(String bankEquId);        //校验
}
```

【代码解析】

- ❑ getBankEquipments()方法用来获取分页对象。
- ❑ deleteBankEquipment()方法用来实现删除银行设备信息。
- ❑ saveBankEquipment()方法用来实现保存银行设备信息。
- ❑ updateBankEquipment ()方法用来实现更新银行设备信息。
- ❑ getBankEquipment()方法用来获取银行设备信息。
- ❑ getBank()方法用来获取银行信息。
- ❑ getAllEquipmentTypes()方法用来获取所有的银行设备信息。
- ❑ getBankEquipments()方法用来获取银行设备信息。
- ❑ checkBankEquId()方法用来验证银行设备 ID 号。

代码 27.68　实现关于 BankEquipment 业务的各种方法类：BankEquipmentServicei.java

```
...
public class BankEquipmentServicei implements BankEquipmentService {
    //创建属性
    private BankEquipmentDao bankEquipmentDao;
    private BankDao bankDao;
    private EquipmentTypeDao equDao;
    public BankEquipmentServicei() {           //构造函数
    }
    // 分页展现所有银行设备明细
    public Pageinfo getBankEquipments(int curpage, String cond) {
        int allrecord = this.getBankEquipmentDao().count(cond);
        List list = this.getBankEquipmentDao().getBankEquipments(curpage,
                pagerecord, cond);
        Pageinfo pageinfo = new Pageinfo(curpage, allrecord, pagerecord,
        list);
        return pageinfo;
    }
    //通过设备流水 ID 删除一个设备明细
    public void deleteBankEquipment(String equipmenteachId) {
        this.getBankEquipmentDao().deleteBankEquip(equipmenteachId);
    }
    //通过设备流水 ID 得到一个设备明细
    public BankEquipment getBankEquipment(String equipmenteachId) {
        BankEquipment bankEquipment = this.getBankEquipmentDao()
                .getBankEquipment(equipmenteachId);
        return bankEquipment;
    }
    public Bank getBank(String bankId) {        //通过银行编号得到某个银行
        Bank bank = this.getBankDao().getBank(bankId);
        return bank;
    }
    //得到所有的设备种类为新增设备明细做准备
    public List getAllEquipmentTypes() {
        List equipmentTypes = this.getEquDao().getAllEquipmentTypes();
        return equipmentTypes;
    }
```

```
    //新增一个银行设备明细
    public void saveBankEquipment(BankEquipment bankEquipment) {
        Bank bank = this.getBankDao().getBank(
                bankEquipment.getBank().getBankId());  //创建银行对象
        bankEquipment.setBank(bank);                   //设置银行对象
        //创建银行设备类型
        Equipmenttype equipmenttype = this.getEquDao().getEquipmenttype(
                bankEquipment.getEquipmenttype().getEquipmentId());
        bankEquipment.setEquipmenttype(equipmenttype);
        this.getBankEquipmentDao().saveBankEquipment(bankEquipment);
    }
    //修改银行设备明细
    public void updateBankEquipment(BankEquipment bankEquipment) {
        this.getBankEquipmentDao().updateBankEquipment(bankEquipment);
    }
    //根据银行网点 ID 和设备 ID 得到银行下某种银行设备的所有设备明细
    public List<BankEquipment> getBankEquipments(String bankId,
            String equipmentId) {
        List bankEquipment = this.getBankEquipmentDao().getBank-
        Equipments(
                bankId, equipmentId);
        return bankEquipment;
    }
    public String checkBankEquId(String bankEquId) {   //校验银行设备
        return this.getBankEquipmentDao().checkBankEquId(bankEquId);
    }
    //省略属性 bankequipmentdao、bankdao 和 equdao 的 set()和 get()方法
...
}
```

注意：在上述代码中，分别实现了 BankEquipmentService 接口中的所有方法。

10. 关于EquipmentType业务方面的接口和类

关于 EquipmentType 操作的接口为 EquipmentTypeService.java，该类的具体内容如代码 27.69 所示。实现该接口的类为 EquipmentTypeServicei.java，该类的具体内容如代码 27.70 所示。

代码 27.69　实现关于 EquipmentType 业务的各种方法接口：EquipmentTypeService.java

```
...
public interface EquipmentTypeService {
    int pagerecord = 10;                          //分页单位
    Pageinfo getEquipmentType(int curpage, String cond);
    //拿到 Equipmenttype 对象的所有数据
    List<Equipmenttype> getAllEquipmentTypes();
    void save(Equipmenttype equipmenttype); //新增 Equipmenttype 的一个对象
    void delete(String equipmentId);       //删除 Equipmenttype 的一个对象
    void update(Equipmenttype equipmenttype);
                                           //修改 Equipmenttype 的一个对象
    //通过 equipmentId 得到一个 Equipmenttype 对象
    Equipmenttype getEquipmenttype(String equipmentId);
    String checkEquipmentId(String equipmentId); //银行设备种类 ID 是否存在
    String checkEquipmentName(String equipmentName);
                                              //银行设备种类名称是否存在
}
```

【代码解析】

- ❑ getEquipmentType()方法用来获取分页对象。
- ❑ getAllEquipmentTypes()方法用来获取所有银行设备类型信息。
- ❑ save()方法用来实现保存银行设备类型信息。
- ❑ delete()方法用来实现删除银行设备类型信息。
- ❑ update()方法用来实现更新银行设备类型信息。
- ❑ getEquipmenttype()方法用来获取银行设备类型信息。
- ❑ checkEquipmentId()方法用来验证银行设备类型的 ID。
- ❑ checkEquipmentName()方法用来验证银行设备类型的名字。

代码 27.70　实现关于 EquipmentType 业务的各种方法类：EquipmentTypeServicei.java

```
...
public class EquipmentTypeServicei implements EquipmentTypeService {
    private EquipmentTypeDao dao;                        //创建属性
    public EquipmentTypeServicei() {                     //构造函数
    }
    //获取银行设备类型分页对象
    public Pageinfo getEquipmentType(int curpage, String cond) {
        List list = this.getDao().getEquipmentTypes(curpage, pagerecord,
        cond);
        long allrecord = this.getDao().count(cond);
        Pageinfo pageinfo = new Pageinfo(curpage, (int) allrecord,
        pagerecord,
                list);
        return pageinfo;
    }
    //通过一个 Equipmenttype 编号删除一个 Equipmenttype 对象
    public void delete(String equipmentId) {
        try {
            Equipmenttype equipmenttype = this.getEquipmenttype
            (equipmentId);
            this.getDao().delete(equipmenttype);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void save(Equipmenttype equipmenttype) {
                                                         //添加银行设备类型方法
        try {
            this.getDao().save(equipmenttype);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void update(Equipmenttype equipmenttype) {
                                                         //更新银行设备类型方法
        try {
            if (equipmenttype.getEquipmentId() != null) {
                this.getDao().update(equipmenttype);
            }
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
    //拿到所有的 Equipmenttype 对象
    public List<Equipmenttype> getAllEquipmentTypes() {
        List<Equipmenttype> equipmentTypes = this.getDao()
                .getAllEquipmentTypes();
        return equipmentTypes;
    }
    //通过一个编号 equipmentId 拿到一个 Equipmenttype 对象
    public Equipmenttype getEquipmenttype(String equipmentId) {
        Equipmenttype equipmenttype = this.getDao().getEquipmenttype(
                equipmentId);
        return equipmenttype;
    }
    public EquipmentTypeDao getDao() {              //配置属性 dao
        return dao;
    }
    public void setDao(EquipmentTypeDao dao) {
        this.dao = dao;
    }
    //银行设备种类 ID 是否存在
    public String checkEquipmentId(String equipmentId) {
        return this.getDao().checkEquipmentId(equipmentId);
    }
    //银行设备种类名称是否存在
    public String checkEquipmentName(String equipmentName) {
        return this.getDao().checkEquipmentName(equipmentName);
    }
}
```

注意：在上述代码中，分别实现了 EquipmentTypeService 接口中的所有方法。

11．关于FaultRepairType业务方面的接口和类

关于 FaultRepairType 操作的接口为 JobService.java，该类的具体内容如代码 27.71 所示。实现该接口的类为 JobServicei.java，该类的具体内容如代码 27.72 所示。

代码 27.71　实现关于 FaultRepairType 业务的各种方法接口：FaultRepairTypeService.java

```
...
public interface FaultRepairTypeService {
    List<FaultRepairType> getAllFaultRepairTypes();
                                          //获得设备报修问题对象的所有数据
    List getAllFaultRepariTypes(String cond);
    void delete(Long pitypeId);                //删除一个设备报修问题对象
    void update(FaultRepairType frt);          //修改一个设备报修问题对象
    void save(FaultRepairType frt);            //新增一个设备报修问题对象
    //通过 pitypeId 得到一个设备报修问题对象
    FaultRepairType getOneFaultRepairType(Long pitypeId);
    //校验问题名称是否存在
    String checkFaultRepairtTypeName(String pitypeValue);
    Pageinfo getpageinfo(int curpage, String cond);    //分页
    int pagerecord = 10;
}
```

【代码解析】

- getAllFaultRepairTypes()方法用来获取设备报修问题对象。
- getAllFaultRepariTypes()方法用来获取分页中所有银行设备报修问题对象。
- delete()方法用来实现删除银行设备报修问题对象。
- update()方法用来实现更新银行设备报修问题对象。
- save()方法用来实现保存银行设备报修问题对象。
- getOneFaultRepairType()方法用来获取一个银行设备报修问题对象。
- checkFaultRepairtTypeName()方法用来验证银行设备报修问题的名字。
- getpageinfo()方法用来获取分页对象。

代码 27.72　实现关于 FaultRepairType 业务的各种方法类：FaultRepairTypeServicei.java

```
...
public class FaultRepairTypeServicei implements FaultRepairTypeService {
    private FaultRepairTypeDao dao;                        //创建属性 dao
    public FaultRepairTypeServicei() {                     //构造函数
    }
    public FaultRepairTypeDao getDao() {                   //配置属性 dao
        return dao;
    }
    public void setDao(FaultRepairTypeDao dao) {
        this.dao = dao;
    }
    //通过一个设备问题编号删除一个设备报修问题对象
    public void delete(Long pitypeId) {
        FaultRepairType frt = this.getOneFaultRepairType(pitypeId);
        this.getDao().delete(frt);
    }
    //拿到所有的设备报修问题对象
    public List<FaultRepairType> getAllFaultRepairTypes() {
        List<FaultRepairType> FaultRepairTypes = this.getDao()
                .getAllFaultRepairTypes();
        return FaultRepairTypes;
    }
    //通过一个编号 pitypeId 拿到一个设备报修问题对象
    public FaultRepairType getOneFaultRepairType(Long pitypeId) {
        return (FaultRepairType) this.getDao().getOnePo(pitypeId);
    }
    public void save(FaultRepairType frt) {    //新增一个设备报修问题对象
        this.getDao().save(frt);
    }
    public void update(FaultRepairType frt) {  //修改一个设备报修问题对象
        if (frt.getPitypeId() != null) {
            this.getDao().update(frt);
        }
    }
    //校验问题名称是否存在
    public String checkFaultRepairtTypeName(String pitypeValue) {
        return this.getDao().checkFaultRepairtTypeName(pitypeValue);
    }
    public List getAllFaultRepariTypes(String cond) {
        List list = this.getDao().getAllFaultRepariTypes(cond);
        return list;
    }
    //分页展现 Pageinfo
```

```
    public Pageinfo getpageinfo(int curpage, String cond) {
        Long rows = (Long) this.getDao().count(cond);
        int count = rows.intValue();
        List list = this.getDao().getAllFaultRepairTypes(curpage,
        pagerecord,
                cond);
        Pageinfo pageinfo = new Pageinfo(curpage, count, pagerecord, list);
        return pageinfo;
    }
}
```

注意：在上述代码中，分别实现了 FaultRepairTypeService 接口中的所有方法。

27.3.4　系统管理的表示层

在设计实现关于商业银行设备巡检系统中系统管理的表现层时，利用 Struts 2.0 框架来实现页面的跳转，涉及的类如表 27.18 所示。

表 27.18　表示层Action类

	类　名	类　作　用
1	UsersAction	关于用户操作页面跳转
2	JobAction	关于岗位操作页面跳转
3	FunctionsAction	关于程序功能操作页面跳转
4	DepartmentAction	关于部门操作页面跳转
5	LogsAction	关于日记操作页面跳转
6	PIGroupAction	关于巡检组操作页面跳转
7	PIWokerAction	关于巡检工操作页面跳转
8	BankAction	关于银行操作页面跳转
9	BankEquipmenAction	关于银行设备明细操作页面跳转
10	EquipmentTypeAction	关于银行设备类型操作页面跳转
11	FaultRepairTypeAction	关于银行设备报修问题类型操作页面跳转

1．关于用户操作的表示层

关于用户操作的所有请求都由 Struts 2.0 框架中名为 UsersAction 的 Action 类来处理，该类的具体内容如代码 27.73 所示。

代码 27.73　关于用户操作的跳转：UsersAction.java

```
...
public class UsersAction extends BaseAction {
    //创建属性
    private UsersService usersService;
    private Users users;
    private Pageinfo pageinfo = new Pageinfo();
    private List<Job> jobs;
    private List<Department> departments;
    private List<Xtymb> yms;
```

```
public UsersAction() {                              //构造函数
}
//通过用户名和密码登录系统
public String login() throws Exception {
    users = this.getUsersService().findUser(users.getLoginId(),
            users.getLoginPassword());
    //为页面响应脚本提示做准备
    this.getResponse().setContentType("text/html;charset=utf-8");
    PrintWriter out = this.getResponse().getWriter();
    if (users != null) {
        if ("1".equals(users.getUserStatus().trim())) {
            this.getSession().setAttribute("user", users);
            //日志记录
            Logs log = new Logs();
            log.setUsersId(users.getLoginId());
            log.setCheckinTime(new Date());
            Serializable logId = this.getUsersService().saveLogs(log);
            this.getSession().setAttribute("logId", logId);
            this.getResponse().sendRedirect("../xtgl/initdata.do");
                                                    //调用初始化数据
            return null;
        } else {
            out.println("<script type=\"text/javascript\">");
            out.println("alert('你的账号被禁用了，请与管理员联系！');");
            out.println("history.back();");
            out.println("</script>");
            return null;
        }
    } else {
        out.println("<script type=\"text/javascript\">");
        out.println("alert('用户名或密码有误，请重新输入！');");
        out.println("history.back();");
        out.println("</script>");
        return null;
    }
}
//初始化数据
public String initData() {
    users = (Users) this.getSession().getAttribute("user");
    String IP = this.getRequest().getRemoteAddr();
    Bank bank = this.getUsersService().getBank(IP);
    if (bank != null) {
        this.getSession().setAttribute("wdid", bank.getBankId());
    } else {
        this.getSession().setAttribute("wdid", null);
    }
    if (users.getJob() != null && users.getJob().getJobId() > 2000) {
                                                    //如果大于 2000 为巡检工
        this.getSession().setAttribute("gid",
                users.getJob().getPiGroup().getGroupId());
        this.getRequest().setAttribute("url", "../sbbx/sbbx!listg4.
        do");
    } else if (users.getJob() != null && users.getJob().getJobId() < 2000
            && bank != null) {//银行网点工作人员登录
        this.getRequest().setAttribute("url", "../sbbx/sbbx!listw4.
        do");
    } else if (("admin").equals(users.getUserName())) {
                                    //如果用户名为 admin，则为后台管理人员
        this.getRequest().setAttribute("url", "../center.jsp");
```

```
        } else {
            this.getRequest().setAttribute("url", "../sbbx/sbbx!listxj.
            do");//为巡检中心的人
        }
        //根据登录用户对应的岗位，得到该用户可以操作的模块，以及模块下的页面
        List<Functions> mks = this.getUsersService().getUserFun(
                users.getJob().getJobId());
        this.getSession().setAttribute("mks", mks);
        yms = this.getUsersService().getXtyms(users.getJob().getJobId(),
                mks.get(0).getFuncId());
        this.getSession().setAttribute("yms", yms);
        return "success";
    }
    public String logout() {                                //退出功能
        Object logId = this.getSession().getAttribute("logId");
        Logs logs = this.getUsersService().getOneLogs(
                Long.parseLong(logId.toString()));
        if (logs != null) {
            logs.setCheckoutTime(new Date());
            this.getUsersService().updateLogs(logs);
        }
    this.getSession().removeAttribute("user");
        this.getSession().removeAttribute("users");
        return "login";
    }
    public String listUsers() {                             //展现一页的用户数据
        String cond = "";
        //从页面中得到要查询的用户登录 ID
        String username = this.getRequest().getParameter("username");
        //从页面中得到要查询用户的中文名字
        String chname = this.getRequest().getParameter("chname");
        if (username != null && !"".equals(username)) {
            cond += " where A.loginId like '%" + username + "%'";
        }
        if (chname != null && !"".equals(chname)) {
            cond += " where A.userName like '%" + chname + "%'";
        }
        pageinfo = this.getUsersService().findAllUsers(pageinfo.
        getCurpage(),
                cond);
        return "success";
    }
    //获取所有的岗位和部门
    public String presave() {
        jobs = this.getUsersService().getAllJobs();
        departments = this.getUsersService().getAllDepartments();
        return "presave";
    }
    public String saveUsers() {                             //新增用户
        this.getUsersService().save(users);
        return "aftersave";
    }
    //准备修改用户，得到整个用户，并得到所有的部门和岗位
    public String preupdate() {
        jobs = this.getUsersService().getAllJobs();
        departments = this.getUsersService().getAllDepartments();
        users = this.getUsersService().findUsersById(users.getLoginId());
        return "preupdate";
    }
    public String updateUsers() {                           //修改用户
```

```
        this.getUsersService().update(users);
        return "afterupdate";
    }
    public String deleteUsers() {              //删除后直接转向用户展现页面
        this.getUsersService().delete(users.getLoginId());
        return "success";
    }
    public String userupdatestate() {          //更改用户登录状态
        this.getUsersService().updateUsersState(users.getLoginId());
        return "success";
    }
    public String getUserYms() {               //得到用户某个模块的页面
        long funcId = Long.parseLong(this.getRequest().getParameter
        ("funcId"));
        Users user = (Users) this.getSession().getAttribute("user");
        this.getSession().removeAttribute("yms");
        yms  =  this.getUsersService().getXtyms(user.getJob().getJobId(),
            funcId);
        this.getSession().setAttribute("yms", yms);
        return "success";
    }
    public String checkUserNameExist() {    //检查此用户名是否在数据库中存在
        long number = this.getUsersService().checkUserNameExist(
                users.getLoginId());
        try {

    this.getResponse().setContentType("text/html;charset=utf-8");
            PrintWriter out = this.getResponse().getWriter();
            out.print(number);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
    //省略属性 usersservice、users、pageinfo、departments、jobs 和 yms 的 set()
    和 get()方法
...
}
```

接着在 struts.xml 文件中配置关于用户操作的请求。

```
<!--配置名为 userLogin 的 Action-->
<action name="userLogin" method="login" class="usersAction">
    <result name="login" type="redirect">../login.jsp</result>
    <result name="input">../500.jsp</result>
</action>
<!--配置名为 userlist 的 Action-->
<action name="userlist" method="listUsers"
    class="usersAction">
    <result name="success">/xtgl/userlist.jsp</result>
</action>
<!--配置名为 userpresave 的 Action-->
<action name="userpresave" method="presave"
    class="usersAction">
    <result name="presave">/xtgl/usernew.jsp</result>
</action>
<!--配置名为 usersave 的 Action-->
<action name="usersave" method="saveUsers"
    class="usersAction">
    <result name="aftersave" type="redirect">
```

```
        /xtgl/userlist.do
    </result>
</action>
<!--配置名为 userpreupdate 的 Action-->
<action name="userpreupdate" method="preupdate"
    class="usersAction">
    <result name="preupdate">/xtgl/userupdate.jsp</result>
</action>
<!--配置名为 userupdate 的 Action-->
<action name="userupdate" method="updateUsers"
    class="usersAction">
    <result name="afterupdate" type="redirect">
        /xtgl/userlist.do
    </result>
</action>
<!--配置名为 userdelete 的 Action-->
<action name="userdelete" method="deleteUsers"
    class="usersAction">
    <result name="success" type="redirect">
        /xtgl/userlist.do
    </result>
</action>
<!--配置名为 userupdatestate 的 Action-->
<action name="userupdatestate" method="userupdatestate"
    class="usersAction">
    <result name="success" type="redirect">
        /xtgl/userlist.do
    </result>
</action>
<!--配置名为 userym 的 Action-->
<action name="userym" method="getUserYms" class="usersAction">
    <result name="success">../left.jsp</result>
</action>
<!--配置名为 login 的 Action-->
<action name="userlogout" method="logout" class="usersAction">
    <result name="login" type="redirect">../login.jsp</result>
</action>
<!--配置名为 checkUserName 的 Action-->
<action name="checkUserName" method="checkUserNameExist"
    class="usersAction">
</action>
<!--配置名为 initdata 的 Action-->
<action name="initdata" method="initData" class="usersAction">
    <result name="success">../main.jsp</result>
</action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于用户功能涉及的页面有：实现登录功能的页面 login.jsp，具体内容如代码 27.74 所示；实现显示所有用户功能的页面 userlist.jsp，具体内容如代码 27.75 所示；实现增加用户的页面 usernew.jsp，具体内容如代码 27.76 所示；实现修改用户的页面 userupdate.jsp，具体内容如代码 27.77 所示。

代码 27.74　登录页面：login.jsp

```
...
    <body>
...
        <form name="form1" method="post" action="./xtgl/userLogin.do"
        onsubmit="return checkdata();">
```

```
            <table >
...
                    <!--用户名输入框-->
                <td>
                    <input name="users.loginId" type="text" id="textfield">
                </td>
                    <!--密码输入框-->
                <td valign="bottom">
                    <input name="users.loginPassword" type="password">
                </td>
...
            </table>
        </form>
    </body>
...
```

代码 27.75　展示所有用户：userlist.jsp

```
...
        <table width="95%" >
            <!--表格的相关标题-->
            <tr>
                <td>用户登录 ID</td>
                <td>用户中文名字</td>
                <td>用户所在部门</td>
                <td>用户所属岗位</td>
                <td>用户状态</td>
                <td>操作</td>
            </tr>
            <!--遍历结果-->
            <s:iterator value="pageinfo.pagedata" id="user">
                <tr align="center">
                    <td>${user.loginId }</td>      <!--输出用户登录 ID -->
                    <td >${user.userName }</td>    <!--输出用户中文名字-->
                    <td >${user.departmentName }</td>
                                                   <!--输出用户所在部门-->
                    <td >${user.jobName }</td>     <!--输出用户所属岗位-->
                    <!--输出用户状态-->
                    <td >${user.userStatus =='1'? '启用':'禁用' }</td>
                    <td >
                        <!--实现编辑功能-->
                        <a href="../xtgl/userpreupdate.do?users.loginId
                        =${user.loginId }">
                        <img src="../images/fix.gif" title="编辑" border
                        ="0" /></a>
                        <!--实现更改用户状态功能-->
                        <ahref="../xtgl/userupdatestate.do?users.loginId
                        =${user.loginId }">
                        <img src="../images/editstate.gif" title="更改用户
                        状态" border="0"
                            /></a>
                        <s:if test="loginId != 'admin'">
                        <!--实现删除功能-->
                        <a href="../xtgl/userdelete.do?users.loginId=$
                        {user.loginId }"
                            title="删除" onClick="return confirm('确定删除
                            此信息吗？')">
```

```
                    <imgsrc="../images/del.gif" border="0"></a>
                  </s:if>
               </td>
           </tr>
        </s:iterator>
     </table>

     <p>
        <font>共${pageinfo.allrecord}</font>项
        <font >每页${pageinfo.pagerecord}</font>项
        <font >当前第${pageinfo.curpage}</font>页
        <a href="../xtgl/userlist.do?pageinfo.curpage=1">首 页</a>
        <a href="../xtgl/userlist.do?pageinfo.curpage=${pageinfo.
        previouspage}">上一页</a>
        <ahref="../xtgl/userlist.do?pageinfo.curpage=${pageinfo.
        nextpage }">    下一页</a>
        <a href="../xtgl/userlist.do?pageinfo.curpage=${pageinfo.
        allpage}">尾页</a>
        第<input id="pagebox" type="text">页
        <a  onclick="goto('../xtgl/userlist.do?pageinfo.curpage=');"
        href="#">跳转</a>
  </body>
...
```

代码 27.76　实现用户新增页面：usernew.jsp

```
...
     <form>
        <table >
           <tr >
              <!--用户登录 ID 输入框 -->
              <td >用户登录 ID
                 <input type="text" name="users.loginId">
                 (只能是字母和数字长度不能大于 10)
              </td>
              <!--用户登录密码输入框 -->
              <td>用户登录密码
                 <input type="password" name="users.loginPassword" >
              </td>
              <!--确认密码输入框 -->
              <td>确认密码
                 <input type="password" name="checkpwd">
              </td>
              <!--用户中文名称输入框 -->
              <td>用户中文名称
                 <input type="text" name="users.userName">
              </td>
              <!--用户所属部门选择框 -->
              <td>用户所属部门
                 <s:select name="users.department.departmentId"
                    list="#request.departments" listKey="
                    departmentId"
                    listValue="departmentName" theme="simple">
                 </s:select>
              </td>
              <!--用户所在岗位选择框 -->
              <td>用户所在岗位
```

```
                    <s:select name="users.job.jobId" list="#request.
                    jobs"
                        listKey="jobId" listValue="name" theme=
                        "simple"></s:select>
                </td>
                <!--用户状态选择框 -->
                <td>用户状态
                    <input type="radio" name="users.userStatus" value
                    ="1" checked>
                    启用
                    <input type="radio" name="users.userStatus" value
                    ="0">
                    禁用
                </td>
                <!--新增和取消按钮-->
                <td>
                    <input type="submit" value="新增">
                    <input type="button" value="取消" onclick="history.
                    back();">
                </td>
        </form>
...
```

代码 27.77　实现用户修改页面：userupdate.jsp

```
...
        <form action="../xtgl/userupdate.do">
            <table >
                <caption >
                    用户修改
                </caption>
                <tr>
                    <input type="hidden" name="users.loginId" value="${
                    users.loginId }">
                    <td>用户登录 ID${users.loginId }</td>
                    <!--用户登录密码输入框-->
                    <td>用户登录密码
                    <input type="password" name="users.loginPassword"
                        value="${users.loginPassword }">
                    </td>
                    <!--确认密码输入框-->
                    <td>确认密码
                        <input type="password" name="checkpwd" class=
                        "input"
                            value="${users.loginPassword }">
                    </td>
                    <!--用户中文名称输入框-->
                    <td>用户中文名称
                        <input type="text" name="users.userName"
                            value="${users.userName }">
                    </td>
                    <!--用户所属部门选择框-->
                    <td>用户所属部门
                        <s:select name="users.department.departmentId"
                            list="#request.departments" listKey="
                            departmentId"
                            listValue="departmentName" theme="simple"><
                            /s:select>
```

```
                    </td>
                    <!--用户所在岗位选择框-->
                    <td>用户所在岗位
                        <s:select name="users.job.jobId" list="#request.
                        jobs"
                            listKey="jobId" listValue="name" theme="simp-
                            le"></s:select>
                    </td>
                    <!--用户状态单选按钮-->
                    <td>用户状态
                        <input type="radio" name="users.userStatus" value
                        ="1"
                            ${users.userStatus=='1'?'checked':'' }>启用
                        <input type="radio" name="users.userStatus" value
                        ="0"
                            ${users.userStatus=='0'?'checked':'' }>禁用
                    </td>
                    <!--修改和返回按钮-->
                    <td>
                        <input type="submit" value="修改">
                        <input type="button" value="返回" onclick="history.
                        back();">
                    </td>
        </form>>
...
```

2. 关于岗位操作的表示层

关于岗位操作的所有请求，都由 Struts 2.0 框架中名为 JobAction 的 Action 类来处理，该类的具体内容如代码 27.78 所示。

代码 27.78　关于岗位操作的跳转：JobAction.java

```
...
public class JobAction extends BaseAction {
    //创建字段
    private Pageinfo pageinfo = new Pageinfo();
    private Job job;
    private JobService service;
    private FunctionService funService;
    private List<Xtymb> list;
    private Functions function;
    private List<Functions> funs;
    private List ids;
    public String list() {                              //展现 Job
        String first = this.getRequest().getParameter("first");
        if ("1".equals(first)) {
            pageinfo.setCurpage(1);
        }
        pageinfo = this.getService().getPageData(pageinfo.getCurpage(),
            "", "");
        return "success";
    }
    public String save() {                              //保存 Job
        String gwxz = this.getRequest().getParameter("gwxz");
        this.service.saveJob(job, gwxz);
        return "save";
    }
    public String preupdate() {                         //准备修改 Job
```

```
        job = this.getService().getOneJobById(job.getJobId());
        return "preupdate";
    }
    public String update() {                              //修改 Job
        this.getService().updateJob(job);
        return "update";
    }
    public String delete() {                              //删除 Job
        this.getService().deleteOneJobById(job.getJobId());
        return "delete";
    }
    public String addGzz() {                              //添加巡检组
        job = this.service.getOneJobById(job.getJobId());
        list = this.service.getAllGroups();
        return "success";
    }
    public String editGroup() {                           //编辑巡检组
        this.service.editGroup(job);
        return "success";
    }
    public String viewXtyms() {                           //展现 Xtymb
        list = this.service.getXtymsByFunctions(function);
        ids = this.service.getXtymsByJobIdAndFuncId(job, function);
        function = this.getFunService().getOneFuntionById(function.
        getFuncId());
        return "view";
    }
    public String viewFunctions() {                       //展现主要功能模块
        funs = this.service.getAllFuns();
        return "success";
    }
    public String addYmToGw() {                           //给岗位添加页面数据
        String[] ymbhs = this.getRequest().getParameterValues("ymbhs");
        list = this.service.addYmtoGw(ymbhs, job, function);
        return "success";
    }
    public String jobshow() {
        pageinfo = this.getService().getAllYmForJob(job.getJobId(),
                pageinfo.getCurpage());
        return "success";
    }
    public String checkJobName() {
        String name = "name";
        job = this.getService().getOneJobByJoName(name, job.getName());
        String i = "";
        if (job == null) {
            i = "0";
        } else {
            i = "1";
        }
        try {
            PrintWriter out = this.getResponse().getWriter();
            out.print(i);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
    //省略属性 job、pageinfo、service、list、funs、function、ids 和 funservice
    的 get()和 set()方法
```

```
...
}
```

接着在 struts.xml 文件中配置关于岗位操作的请求。

```
<!--配置名为 joblist 的 Action-->
<action name="joblist" method="list" class="jobAction">
    <result name="success">./joblist.jsp</result>
</action>
<!--配置名为 jobsave 的 Action-->
<action name="jobsave" method="save" class="jobAction">
    <result name="save" type="redirect">./joblist.do</result>
</action>
<!--配置名为 jobdelete 的 Action-->
<action name="jobdelete" method="delete" class="jobAction">
    <result name="delete" type="redirect">./joblist.do</result>
</action>
<!--配置名为 jobpreupdate 的 Action-->
<action name="jobpreupdate" method="preupdate"
    class="jobAction">
    <result name="preupdate">./jobupdate.jsp</result>
</action>
<!--配置名为 jobupdate 的 Action-->
<action name="jobupdate" method="update" class="jobAction">
    <result name="update" type="redirect">./joblist.do</result>
</action>
<!--配置名为 addGzz 的 Action-->
<action name="addGzz" method="addGzz" class="jobAction">
    <result name="success">./addGroup.jsp</result>
</action>
<!--配置名为 editGroup 的 Action-->
<action name="editGroup" method="editGroup" class="jobAction">
    <result name="success" type="redirect">./joblist.do</result>
</action>
<!--配置名为 view 的 Action-->
<action name="view" method="viewXtyms" class="jobAction">
    <result name="view">./xtym.jsp</result>
</action>
<!--配置名为 viewFun 的 Action-->
<action name="viewFun" method="viewFunctions"
    class="jobAction">
    <result name="success">./functionlist.jsp</result>
</action>
<!--配置名为 addYmToGw 的 Action-->
<action name="addYmToGw" method="addYmToGw" class="jobAction">
    <result name="success">./success.jsp</result>
</action>
<!--配置名为 checkJobName 的 Action-->
<action name="checkJobName" method="checkJobName"
    class="jobAction">
</action>
<!--配置名为 jobshow 的 Action-->
<action name="jobshow" method="jobshow" class="jobAction">
    <result name="success">./showxtym.jsp</result>
</action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于用户功能涉及的页面有：实现显示所有岗位功能的页面 joblist.jsp，具体内容如代码 27.79 所示；实现增加岗位的页面

usernew.jsp，具体内容如代码 27.80 所示；实现修改岗位的页面 jobupdate.jsp，具体内容如代码 27.81 所示；实现显示所有程序功能的页面 functionlist.jsp，具体内容如代码 27.82 所示；实现绑定程序功能中各个子功能的页面 xtym.jsp，具体内容如代码 27.83 所示；实现绑定巡检组的页面 addGroup.jsp，具体内容如代码 27.84 所示。

代码 27.79　显示岗位列表页面：joblist.jsp

```
...
    <table>
        <tr>
            <td>
                <!--新增按钮 -->
                <input name="button" type="image" value="新增 href='./
                jobnew.jsp'" />
            </td>
...
            <!--表格的相关标题-->
            <td bgcolor="#F2F2F2" align="center" width="15%">编号</td>
            <td bgcolor="#F2F2F2" align="center" width="15%">名称</td>
            <td bgcolor="#F2F2F2" align="center" width="15%">描述</td>
            <td bgcolor="#F2F2F2" align="center" width="15%">操作</td></tr>
            <!--遍历岗位结果-->
            <s:iterator value="pageinfo.pagedata">
...
            <!--实现功能-->
            <a href="./jobshow.do?job.jobId=${jobId }&pageinfo.curpage=1"
                    title="查看拥有的权限">${name}</a>
            </td>
            <td>${description}
            <!--实现编辑功能-->
            <a href="./jobpreupdate.do?job.jobId=${jobId }"><img
                src="../images/fix.gif" title="编辑" border="0"></a>
            </td>
            <td>${description}
            <!--实现模块管理功能-->
            <a href="./viewFun.do?job.jobId=${jobId }"><img
                src="../images/VIEW.GIF" title="模块管理" border="0"></a>
            <s:if test="%{#userses == null && userses.size()==0}">
            <!--实现删除功能-->
                <a href="./jobdelete.do?job.jobId=${jobId }"
                    onClick="return confirm('确定删除此信息吗？')">
                    <img src="../images/del.gif" title="删除" border="0">
                </a>
            </s:if>
            <!--实现编辑工作组功能-->
            <s:if test="jobId>2000">
                <a href="./addGzz.do?job.jobId=${jobId }">
                    <img title="编辑工作组" border="0">
                </a>
            </s:if>
            </td>
...
            <!--实现分页结果-->
            <td>
            <font>共${pageinfo.allrecord}项</font>
```

```
            <font>每页${pageinfo.pagerecord}项</font>
            <font>当前第${pageinfo.curpage}页</font>
            <font>共${pageinfo.allpage}页</font>
            <s:if test="%{pageinfo.curpage!=1}">
            <a href="./joblist.do?pageinfo.curpage=1">首页</a>
            <a href="./joblist.do?pageinfo.curpage=${pageinfo.Previousp-
            age }">上一页</a>
            </s:if>
            <s:if test="%{pageinfo.curpage!=pageinfo.allpage}">
            <a href="./joblist.do?pageinfo.curpage=${pageinfo.nextpage
            }">下一页</a>
            <a href="./joblist.do?pageinfo.curpage=${pageinfo.allpage }">
            尾页</a>
            </s:if>
            <input id="pagebox" size="1" name="pagebox"></input>
            <a onclick="goto('./joblist.do?m=list&');" href="#">跳转
            <input id="totalpage" type="hidden" size="1" value="$
            {pageinfo.allpage }"
                name="totalpage"></input>
            </td>
...
```

代码 27.80　实现增加岗位页面：jobnew.jsp

```
...
        <form action="${url }" method="post">
            <table>
                <tr>
                    <td>
                        <!--岗位性质选择框-->
                        岗位性质
                        <input type="radio" name="gwxz" value="gl" checked
                        ="checked">
                        <!--管理单选按钮-->
                        管理
                        <input type="radio" name="gwxz" value="xjz">
                        巡检组
                    </td>
                </tr>
                <tr>
                    <!--岗位名称输入框-->
                    <td>
                        岗位名称:
                        <input name="job.name" id="jobName" >
                    </td>
                </tr>
                <tr>
                    <!--岗位描述框-->
                    <td>
                        岗位描述:
                        <textarea name="job.description" cols="30" checked
                        X></textarea>
                    </td>
                </tr>
            </table>
            <center>
                <!--新增和返回按钮 -->
```

```
                <input type="submit" value="新增">
                <input type="button" value="返回" onclick="history.back
                ();">
            </center>
        </form>
...
```

代码 27.81　实现修改岗位页面：jobupdate.jsp

```
...
        <form action="${url }" method="post">
            <table >
                <input name="job.jobId " value="${job.jobId }" type="hidden">
                <tr>
                    <!--岗位名称输入框 -->
                    <td >岗位名称:
                        <input name="job.name" value="${job.name }" id=
                        "jobName">
                    </td>
                </tr>
                <tr>
                    <!--岗位描述输入框 -->
                    <td >岗位描述:
                        <textarea name="job.description">${job.Descrip-
                        tion}</textarea>
                    </td>
                </tr>
            </table>
            <center>
                <!--修改和返回按钮 -->
                <input type="submit" value="修改">
                <input type="button" value="返回" onclick="history.back
                ();">
            </center>
        </form>
...
```

代码 27.82　实现修改岗位页面：functionlist.jsp

```
...
     <table>
        <tr>
            <!--表格的相关标题-->
            <td>编号</td>
            <td>名称</td>
            <td>操作</td>
        </tr>
        <!-- 遍历岗位结果-->
        <s:iterator value="funs">
            <tr >
                <td>
                    ${funcId }                  <!--输出编号-->
                    ${funcName } </td>          <!--输出名称-->
                    <!--实现页面列表功能-->
                    <a href="./view.do?function.funcId=${funcId}&job.
                    jobId=${job.jobId  }"><img  src="../images/VIEW.GIF"
                        border="0" title="页面列表"></a>
```

```
                </td>
            </tr>
        </s:iterator>
    </table>
...
```

代码 27.83　绑定功能页面：xtym.jsp

```
...
        <form action="${url }" method="post">
                <table >
                    <div>
                    <!--复选框-->
                    <input type="checkbox" name="allbox">全选/撤选</div>
                    <!--表格的相关标题-->
                    <td >选择</td>
                    <td>页面名称</td>
                    <!--遍历页面名称结果-->
                    <s:iterator value="list" id="xtymb" status="i">
                        <s:if test="%{#xtymb.ymbh in ids}">
                            <input type="checkbox" value="${xtymb.ymbh }"
                            checked
                                    name="ymbhs">${xtymb.ymmc }
                        </s:if>
                        <s:else>
                            <input type="checkbox" value="${xtymb.ymbh }"
                                    name="ymbhs">${xtymb.ymmc }
                        </s:else>
                    </s:iterator>
                </table>
            <!--新增和返回按钮-->
            <center>
                <input type="submit" value="新增">
                <input type="button" value="返回" onclick="history.back
                ();">
            </center>
        </form>
...
```

代码 27.84　实现绑定巡检组页面：addGroup.jsp

```
...
        <form action="${url }" method="post">
            <table>
                <tr>
                    <!--表格的相关标题-->
                    <td>
                        岗位性质
                    </td>
                    <td>
                        巡检组
                    </td>
...
                    <!--输出岗位信息-->
                    <td>
                        岗位名称:
                        ${job.name}
```

```
                    <input type="hidden" name="job.jobId" value="$
                    {job.jobId }">
                </td>
...
                <!--表格的相关标题-->
                <td>
                    选择
                </td>
                <td>
                    巡检组编号
                </td>
                <td>
                    巡检组名称
                </td></tr>
            <!--遍历巡检组结果-->
            <s:iterator value="list" id="group">
                    <td>
                        <!--单选按钮-->
                        <s:if test="%{#group.groupId == job.piGroup.
                        groupId}">
                            <input type="radio" name="job.piGroup.
                            groupId"
                                value="${group.groupId }" checked>
                        </s:if><s:else>
                            <input type="radio" name="job.piGroup.
                            groupId"
                                value="${group.groupId }">
                        </s:else></td>
                <!--输出巡检组编号-->
                    <td>
                        ${group.groupId }
                    </td>
                <!--输出巡检组名称-->
                    <td>
                        ${group.groupName }
                    </td>
            </s:iterator>
        </table>
        <center>
            <!--提交和取消按钮-->
            <input type="submit" value="提交">
            <input type="button" value="取消" onclick="history.back
            ();">
        </center>
    </form>
...
```

3. 关于程序功能操作的表示层

关于程序功能操作的所有请求，都由 Struts 2.0 框架中名为 FunctionAction 的 Action 类来处理，该类的具体内容如代码 27.85 所示。

代码 27.85　关于程序功能的跳转：FunctionAction.java

```
...
public class FunctionAction extends BaseAction {
    //创建属性
```

```
    private FunctionService service;
    private Pageinfo pageinfo;
    private Functions function;
    private Job job;
    public String list() {                          //展现 Functions
        pageinfo = this.getService().getPageData(1, "", "");
        return "success";
    }
    public String save() {                          //保存 Functions
        this.service.saveFunction(function);
        return "save";
    }
    public String preupdate() {                     //准备修改 Functions
        function = this.getService().getOneFuntionById(function.
        getFuncId());
        return "preupdate";
    }
    public String update() {                        //修改 Functions
        this.getService().update(function);
        return "update";
    }
    public String delete() {                        //删除 Functions
        System.out.println(function.getFuncId() + "-------------");
        this.getService().deleteOneFunctionById(function.getFuncId());
        return "delete";
    }
    //省略属性 function、pageinfo、service 和 job 的 get()和 set()方法
...
}
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!-- 程序功能 FunctionAction 的配置 -->
<action name="funlist" method="list" class="funAction">
    <result name="success">/xtgl/functionlist.jsp</result>
</action>
<!--配置名为 funsave 的 Action-->
<action name="funsave" method="save" class="funAction">
    <result name="save" type="redirect">./funlist.do</result>
    <!-- 使用转向，避免重复提交 -->
</action>
<!--配置名为 fundelete 的 Action-->
<action name="fundelete" method="delete" class="funAction">
    <result name="delete" type="redirect">./funlist.do</result>
</action>
<!--配置名为 funpreupdate 的 Action-->
<action name="funpreupdate" method="preupdate"
    class="funAction">
    <result name="preupdate">./functionupdate.jsp</result>
</action>
<!--配置名为 funupdate 的 Action-->
<action name="funupdate" method="update" class="funAction">
    <result name="update" type="redirect">./funlist.do</result>
</action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于用户功能涉及的页面有：实现增加程序功能的页面 functionnew.jsp，具体内容如代码 27.86 所示；实现修改程序功能的页面 functionupdate.jsp，具体内容如代码 27.87 所示。

代码 27.86　实现添加程序功能页面：functionnew.jsp

```
...
    <form action="${url }" method="post">
      <table>
      <!--程序功能名称输入框 -->
        <tr>
          <td>程序功能名称</td><td><input name="function.funcName"></td>
        </tr>
      </table>
      <!--提交和取消按钮 -->
      <center><input type="submit" value="提交">
      <input type="button" value="取消" onclick="history.back();"></
      center>
    </form>
...
```

代码 27.87　实现程序功能页面：functionupdate.jsp

```
...
    <form action="${url }" method="post">
      <table>
      <!--程序功能名称输入框 -->
        <tr>
          <td>程序功能名称</td><td><input name="function.funcName" value
          ="${function.funcName }"></td>
        </tr>
      </table>
      <!--提交和取消按钮 -->
      <center><input type="submit" value="提交">
      <input type="button" value="取消" onclick="history.back();"></
      center>
    </form>
...
```

4．关于部门操作的表示层

关于部门操作的所有请求，都由 Struts 2.0 框架中名为 DepartmentAction 的 Action 类来处理，该类的具体内容如代码 27.88 所示。

代码 27.88　关于部门操作的跳转：DepartmentAction.java

```
...
public class DepartmentAction extends BaseAction {
    //创建属性
    private DepartmentService service;
    private Pageinfo pageinfo = new Pageinfo();
    private Department dept;
    //省略属性 dept、service 和 pageinfo 的 get()和 set()方法
...
    public String list() {                                  //分页展现
        pageinfo = this.getService().getPage(pageinfo.getCurpage(), "");
        return "success";
    }
    public String save() {                                  //保存
        this.getService().saveOne(dept);
```

```
        return "success";
    }
    public String delete() {                            //删除
        this.getService().deleteOne(dept);
        return "success";
    }
    public String preupdate() {                         //准备修改
        dept = this.getService().getOne(dept.getDepartmentId());
        return "update";
    }
    public String update() {                            //更新数据
        this.getService().updateOne(dept);
        return "success";
    }
    public String checkDeptname() {                     //校验部门名称
        //获取所需校验部门名称
        String name = this.getRequest().getParameter("name");
        String rs = this.getService().checkDeptname(name);
        try {
            this.getResponse().getWriter().println(rs);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!-- 部门展现 -->
<action name="deptlist" method="list" class="deptAction">
    <result name="success">/xtgl/deptlist.jsp</result>
</action>
<!--配置名为 deptsave 的 Action-->
<action name="deptsave" method="save" class="deptAction">
    <result name="success" type="redirect">
        /xtgl/deptlist.do
    </result>
</action>
<!--配置名为 deptdelete 的 Action-->
<action name="deptdelete" method="delete" class="deptAction">
    <result name="success" type="redirect">
        /xtgl/deptlist.do
    </result>
</action>
<!--配置名为 deptpreupdate 的 Action-->
<action name="deptpreupdate" method="preupdate"
    class="deptAction">
    <result name="update">/xtgl/deptupdate.jsp</result>
</action>
<!--配置名为 deptupdate 的 Action-->
<action name="deptupdate" method="update" class="deptAction">
    <result name="success" type="redirect">
        /xtgl/deptlist.do
    </result>
</action>
<!--配置名为 checkDeptname 的 Action-->
<action name="checkDeptname" method="checkDeptname"
    class="deptAction">
</action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于用户功能涉及的页面有：实现显示部门的页面 deptlist.jsp，具体内容如代码 27.89 所示；实现增加部门的页面 deptnew.jsp，具体内容如代码 27.90 所示；实现修改部门的页面 deptupdate.jsp，具体内容如代码 27.91 所示。

代码 27.89　实现显示部门页面：deptlist.jsp

```
...
    <table>
            <tr>
                <!--新增按钮-->
                <td>
                    <input name="button" type="image" value="新增" href='.
                    /deptnew.jsp'" />
                </td>
                <!--表格的相关标题-->
                <td >部门编号</td>
                <td >部门名称</td>
                <td >操作</td>
                <!--遍历结果-->
                <s:iterator value="pageinfo.pagedata" id="dept">
                    <td>${departmentId}</td>       <!--输出部门编号-->
                    <td>${departmentName}</td>     <!--输出部门名称-->
                    <!--实现删除功能-->
                    <s:url id="delete" value="/xtgl/deptdelete.do"
                        includeParams="false"></s:url>
                    <s:if test="%{#dept.userses==null || #dept.userses.
                    size()==0}">
                        <a href="${delete }?dept.departmentId=${depar-
                        tmentId }">
                        <img title="删除"onClick="return confirm('确定删除
                        此信息吗？')"
                            ></a>
                    </s:if>
                    <!--实现编辑功能-->
                    <s:url id="preupdate" value="/xtgl/deptpreupdate.do"
                        includeParams="false">
                    </s:url>
                    <a href="${preupdate }?dept.departmentId=${
                    departmentId }">
                    <img title="编辑" border="0"></a>
                </s:iterator>
                <!--关于分页功能-->
                <td>
                    <font>共${pageinfo.allrecord}项</font>
                    <font>每页${pageinfo.pagerecord}项</font>
                    <font>当前第${pageinfo.curpage}页</font>
                    <font>共${pageinfo.allpage}页</font>
                    <s:url id="list"value="/xtgl/deptlist.do" includePa-
                    rams="false"></s:url>
                    <s:if test="%{pageinfo.curpage!=1}">
                    <a href="${list }?pageinfo.curpage=1">首页</a>
                    <a href="${list }?pageinfo.curpage=${pageinfo.
```

```
                    curpage-1 }">上一页</a>
                    </s:if>
                    <s:if test="%{pageinfo.curpage!=pageinfo.allpage}">
                    <ahref="${list }?pageinfo.curpage=${pageinfo.
                    curpage+1 }">下一页</a>
                    <a href="${list }?pageinfo.curpage=${pageinfo.
                    allpage }">尾页</a>
                    </s:if>
                    <input id="pagebox" size="1" name="pagebox"></input>
                    <a onclick="goto('./deptlist.do?m=list&');" href="#">
                    跳转</a>
                    <input id="totalpage" type="hidden" value="${pageinfo.
                    allpage }"
                            name="totalpage"></input>
                </td>
            </tr>
    </table>
...
```

代码 27.90　实现添加部分页面：deptnew.jsp

```
...
        <form action="${save }" method="post">
                    <!--部门名称输入框 -->
                    <td>部门名称</td>
                    <input type="text" name="dept.departmentName" id="
                    deptname"
                        onblur="checkDeptname('deptname')">
                    <!--保存和返回按钮 -->
                    <input type="submit" value="保存">
                    <input type="button" value="返回" onclick="history.back
                    ();">
        </form>
...
```

代码 27.91　实现更新部分页面：deptupdate.jsp

```
...
        <form action="${update }" method="post">
            <table>
                <tr>
                    <!--部门编号输入框 -->
                    <td>部门编号</td>
                    <input name="dept.departmentId" readonly="true"
                        value="${dept.departmentId }"></td>
                    <!--部门名称输入框 -->
                    <td>部门名称</td>
                    <input type="text" name="dept.departmentName"
                        value="${dept.departmentName }">
                    <!--提交和返回按钮 -->
                    <input type="submit" value="修改">
                    <input type="button" value="返回" onclick="history.back
                    ();">
        </form>
...
```

5. 关于日记操作的表示层

关于日记操作的所有请求，都由 Struts 2.0 框架中名为 LogsAction 的 Action 类来处理，该类的具体内容如代码 27.92 所示。

代码 27.92　关于日记操作的跳转：LogsAction.java

```
...
public class LogsAction extends BaseAction {
    //创建属性
    private Pageinfo pageinfo = new Pageinfo();
    private LogsService service;
    private Logs log;
    //省略属性 log、pageinfo 和 service 的 get()和 set()方法
    ...
    public String list()                                      //分页展现
        pageinfo = this.getService().getPage(pageinfo.getCurpage(), "");
        return "success";
    }
    public String deleteAll() {                               //清空日志
        this.getService().deleteAll();
        return this.list();
    }
    public String deleteOne() {                               //删除一条日志
        this.getService().deleteOne(log.getLogId());
        return this.list();
    }
    public String createAllToExcel() {                        //生产 Excel 表格
        List list = this.getService().getAll();
        this.getService().createExcelForAll(list, this.getRequest(),
                this.getResponse());
        return null;
    }
}
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 logslist 的 Action-->
<action name="logslist" method="list" class="logAction">
    <result name="success">/xtgl/logslist.jsp</result>
</action>
<!--配置名为 logsdeleteAll 的 Action-->
<action name="logsdeleteAll" method="deleteAll"
    class="logAction">
    <result name="success">/xtgl/logslist.jsp</result>
</action>
<!--配置名为 logsdeleteOne 的 Action-->
<action name="logsdeleteOne" method="deleteOne"
    class="logAction">
    <result name="success">/xtgl/logslist.jsp</result>
</action>
<!--配置名为 logsAllToExcel 的 Action-->
<action name="logsAllToExcel" method="createAllToExcel"
    class="logAction">
    <result name="success">/xtgl/logslist.jsp</result>
</action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于日记功能涉及的页面有实现显示日记的页面 logslist.jsp，具体内容如代码 27.93 所示。

代码 27.93　实现显示日记记录页面：logslist.jsp

```
...
    <table>
        <!--清空日志按钮-->
        <s:url id="deleteAll" value="/xtgl/logsdeleteAll.do" includee-
        Params="false"></s:url>
        <a href="${deleteAll }">清空日志</a>
        <!--生成 Excel 按钮-->
        <s:url id="createExcel" value="/xtgl/logsAllToExcel.do"include-
        Params="false"></s:url>
        <a href="${createExcel }">生成 excel</a>
        <!--表格的相关标题-->
        <td>日志编号</td>
        <td>登录 ID</td>
        <td>登录时间</td>
        <td>退出时间</td>
        <!--遍历结果-->
        <s:iterator value="pageinfo.pagedata">
            <td>${logId }</td>                    <!--输出日志编号-->
            <td>${usersId }</td>                  <!--输出登录 ID -->
            <!--输出登录时间-->
            <s:date name="checkinTime" format="yyyy-MM-dd HH:MM:ss" /></td>
            <!--输出登出时间-->
            <s:date name="checkoutTime" format="yyyy-MM-dd HH:MM:ss" /></td>
        </s:iterator>
        <!--关于分页功能-->
        <td>
            <font>共${pageinfo.allrecord}项</font>
            <font>每页${pageinfo.pagerecord}项</font>
            <font>当前第${pageinfo.curpage}页</font>
            <font>共${pageinfo.allpage}页</font>
            <s:url id="list"value="/xtgl/deptlist.do" includeParams=
            "false"></s:url>
            <s:if test="%{pageinfo.curpage!=1}">
            <a href="${list }?pageinfo.curpage=1">首页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.curpage-1 }">上
            一页</a>
            </s:if>
            <s:if test="%{pageinfo.curpage!=pageinfo.allpage}">
            <a href="${list }?pageinfo.curpage=${pageinfo.curpage+1 }">下
            一页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.allpage }">尾页
            </a>
            </s:if>
            <input id="pagebox" size="1" name="pagebox"></input>
            <a onclick="goto('./deptlist.do?m=list&');" href="#">跳转</a>
            <input id="totalpage" type="hidden" value="${pageinfo.allpage }"
                name="totalpage"></input>
        </td>
    </table>
...
```

6. 关于巡检组操作的表示层

关于巡检组操作的所有请求，都由 Struts 2.0 框架中名为 PiGroupActio 的 Action 类来处理，该类的具体内容如代码 27.94 所示。

代码 27.94　关于巡检组操作的跳转：PiGroupAction.java

```
...
public class PiGroupAction extends BaseAction {
    //创建属性
    private PiGroupService service;
    private Pageinfo pageinfo = new Pageinfo();
    private PiGroup pigroup;
    private List works;
    private List groups;
    private Long[] sel2;
    private List jworks;
    public PiGroupAction() {                          //构造函数
    }
    public String PiGrouplist() {                     //展现巡检组
        String cond = "";
        pageinfo  =  this.getService().getPiGroups(pageinfo.getCurpage(),
            cond);
        return "PiGrouplistsuccess";
    }
    public String presaveGroup() {                    //准备新增巡检组、巡检工
        works = this.getService().getAllWorks();
        return "save";
    }
    public String savePiGroup() {                     //新增巡检组
        this.getService().savePiGroup(pigroup, sel2);
        return "PiGrouplist";
    }
    public String preupdate() {                       //准备修改巡检组
        pigroup = this.getService().getOnePiGroup(pigroup.getGroupId());
        return "updatePiGroup";
    }
    public String updatePiGroup() {                   //更新巡检组
        this.getService().updatePiGroup(pigroup);
        return "PiGrouplist";
    }
    public String deletePiGroup() {                   //删除巡检组
        this.getService().deletePiGroup(pigroup.getGroupId());
        return "PiGrouplist";
    }
    public String editGroupAndWork() {                //重新分配巡检工
        if (pigroup != null) {
            works = this.getService().getWorksNotinGroup(pigroup.
            getGroupId());
            jworks = this.getService().getWorks(pigroup.getGroupId());
        } else {
            works = this.getService().getAllWorks();
        }
        groups = this.getService().getAllPiGroups();
        return "editWork";
    }
    //省略属性 service、pageinfo、pigroup、groups、works、ser2 和 jworks 的 get()
```

```
    和 set() 方法
    ...
}
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!-- 展现巡检组 -->
<action name="PiGrouplist" method="PiGrouplist" class=
"pigroupAction">
  <result name="PiGrouplistsuccess">/xtgl/pigroup.jsp</result>
</action>
<!--配置名为 presaveGroup 的 Action-->
<action name="presaveGroup" method="presaveGroup" class=
"pigroupAction">
  <result name="save">/xtgl/pigroupnew.jsp</result>
</action>
<!--配置名为 savePiGroup 的 Action-->
<action name="savePiGroup" method="savePiGroup" class=
"pigroupAction">
  <result name="PiGrouplist" type="redirect">/xtgl/PiGrouplist.
  do</result>
</action>
<!--配置名为 preupdatepi 的 Action-->
<action name="preupdatepi" method="preupdate" class="pigroup-
Action">
 <result name="updatePiGroup">/xtgl/pigroupupdate.jsp</result>
</action>
<!--配置名为 updatePiGroup 的 Action-->
<action name="updatePiGroup" method="updatePiGroup" class=
"pigroupAction">
 <result name="PiGrouplist" type="redirect">/xtgl/PiGrouplist.
 do</result>
</action>
<!--配置名为 deletePiGroup 的 Action-->
 <action name="deletePiGroup" method="deletePiGroup" class=
 "pigroupAction">
 <result name="PiGrouplist" type="redirect">/xtgl/PiGrouplist.
 do</result>
</action>
<!--配置名为 editGroupAndWork 的 Action-->
          <action name="editGroupAndWork" method="editGroupAndWork"
          class="pigroupAction">
   <result name="editWork">/xtgl/eiditWork.jsp</result>
</action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于巡检组功能涉及的页面有：实现显示日记的页面 pigroup.jsp，具体内容如代码 27.95 所示；实现增加巡检组的页面 pigroupnew.jsp，具体内容如代码 27.96 所示；实现修改巡检组的页面 pigroupupdate.jsp，具体内容如代码 27.97 所示；实现重新分配巡检工的页面 eiditWork.jsp，具体内容如代码 27.98 所示。

代码 27.95　实现显示巡检组页面：pigroup.jsp

```
...
    <table>
        <!--表格的相关标题-->
        <td>巡检组编号</td>
```

```
        <td>巡检组名称</td>
        <td>操作</td>
        <!--遍历结果-->
        <s:iterator value="pageinfo.pagedata">
            <td>${groupId}</td>                 <!--输出问题编号-->
            <td >${groupName}</td>              <!--输出问题编号-->
            <!--实现修改功能-->
            <a href="./preupdatepi.do?pigroup.groupId=${groupId }">
            <img src="../images/fix.gif" title="修改" border="0"></a>
            <!--实现重新分配功能-->
            <a href="./editGroupAndWork.do?pigroup.groupId=${groupId }">
            重新分配</a>
            <s:if test="%{#piwokers==null&&piwokers.size()==0}">
            <!--实现删除功能-->
            <a href="./deletePiGroup.do?pigroup.groupId=${groupId }"
            onClick="return
                confirm('确定删除此信息吗？')">
            <img src="../images/del.gif" title="删除" border="0"></a>
            </s:if>
            <s:else>
            <!--实现查看巡检功能-->
            <a href="./viewpiwoker1.do?piwoker.piGroup.groupId=$
            {groupId }">查看巡检</a>
            </s:else>
        </s:iterator>
        <!--关于分页功能-->
        <td>
            <font>共${pageinfo.allrecord}项</font>
            <font>每页${pageinfo.pagerecord}项</font>
            <font>当前第${pageinfo.curpage}页</font>
            <font>共${pageinfo.allpage}页</font>
            <s:url id="list"value="/xtgl/deptlist.do" includeParams=
            "false"></s:url>
            <s:if test="%{pageinfo.curpage!=1}">
            <a href="${list }?pageinfo.curpage=1">首页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.curpage-1 }">上
            一页</a>
            </s:if>
            <s:if test="%{pageinfo.curpage!=pageinfo.allpage}">
            <ahref="${list }?pageinfo.curpage=${pageinfo.curpage+1 }">下
            一页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.allpage }">尾页
            </a>
            </s:if>
            <input id="pagebox" size="1" name="pagebox"></input>
            <a onclick="goto('./deptlist.do?m=list&');" href="#">跳转</a>
            <input id="totalpage" type="hidden" value="${pageinfo.allpage }"
                            name="totalpage"></input>
        </td>
    </table>
...
```

代码 27.96　实现新增巡检组页面：pigroupnew.jsp

```
...
        <form action="./updatePiGroup.do" method="post">
```

```
            <table>
                    <!--巡检组编号输入框 -->
                    <td align="center">巡检组编号：</td>
                    <input name="pigroup.groupId" value="${pigroup.
                    groupId}" >
                    <!--巡检组名称输入框 -->
                    <td  align="center">巡检组名称：</td>
                    <input name="pigroup.groupName" value="${pigroup.
                    groupName}">
                    <!--提交和返回按钮 -->
                    <input type="submit" value="提交">
                    <input type="button" value="返回" onclick="history.
                    back();">
            </table>
        </form>
...
```

代码 27.97　实现更新巡检组页面：pigroupupdate.jsp

```
...
        <form action="./updatePiGroup.do" method="post">
            <table>
                    <!--巡检组编号输入框 -->
                    <td align="center">巡检组编号：</td>
                    <input name="pigroup.groupId" value="${pigroup.groupId}" >
                    <!--巡检组名称输入框 -->
                    <td  align="center">巡检组名称：</td>
                    <input name="pigroup.groupName" value="${pigroup.
                    groupName}">
                    <!--提交和返回按钮 -->
                    <input type="submit" value="提交">
                    <input type="button" value="返回" onclick="history.
                    back();">
            </table>
        </form>
...
```

代码 27.98　实现重新分配巡检工页面：eiditWork.jsp

```
...
        <form action="${edit }" method="post">
            <table>
                <!--巡检组选择框 -->
                <td >选择巡检组：
                <select onchange="getChild(this.options[selectedIndex].
                value,'sel2');"
                    id="group" name="group">
                <!--遍历巡检组结果 -->
                <s:iterator value="groups">
                    <option value="${groupId }"
                        <s:if test="%{pigroup.groupId==groupId}">selected
                        ="selected"
                        </s:if>>
                        ${groupName}
                    </option>
                </s:iterator>
                </select>
```

```
                <!--巡检工选择框 -->
                <select id="sel2" name="sel2" multiple="true">
                <!--遍历巡检工结果 -->
                <s:iterator value="jworks">
                    <option value="${wokerId }" selected="true">$
                    {workerName}</option>
                </s:iterator>
                </select>
                <!--实现添加功能 -->
                <input type="button" value="<<" onclick="Add();">
                <!--实现删除功能-->
                <input type="button" value=">>" onclick="Del();">
                <s:select list="works" listKey="wokerId" listValue=
                kerName"
                            theme="simple" multiple="true" name="sel1"
                            id="sel1">
                </s:select>
                <!--提交和返回按钮 -->
                <input type="submit" value="提交">
                <input type="button" value="返回" onclick="history.back
                ();">
            </p>
        </form>
...
```

7. 关于巡检工操作的表示层

关于巡检工操作的所有请求，都由 Struts 2.0 框架中名为 PiwokerAction 的 Action 类来处理，该类的具体内容如代码 27.99 所示。

代码 27.99　关于巡检工操作的跳转：PiwokerAction.java

```
...
public class PiwokerAction extends BaseAction {
    //创建属性
    private PiwokerService service;
    private Pageinfo pageinfo = new Pageinfo();
    private Piwoker piwoker;
    private PiGroup pigroup;
    private Long group;
    public PiwokerAction() {                    //构造函数
    }
    public String Piwokerlist() {               //分页展现巡检工基本信息
        String cond = "";
        pageinfo = this.getService().getPiwokers(pageinfo.getCurpage(),
            cond);
        return "PiwokerlistSuccess";
    }
    public String viewpiwoker() {               //添加巡检组下的巡检工信息
        String[] sel2 = this.getRequest().getParameterValues("sel2");
        this.getService().updateGroupWorker(group, sel2);
        return "PiGrouplist";
    }
    public String viewpiwoker1() {              //查看巡检组下的巡检工信息
        String cond = " and A.piGroup.id= '"
                + piwoker.getPiGroup().getGroupId() + "'";
        pageinfo = this.getService().getPiwokers(pageinfo.getCurpage(),
```

```
            cond);
        pigroup = this.getService().getOnePiGroup(
                piwoker.getPiGroup().getGroupId());
        return "success";
    }
    public String presavePiwoker() {           //准备新增巡检工
        List pigroups = this.getService().getAllPiGroups();
        this.getRequest().setAttribute("pigroups", pigroups);
        return "success";
    }
    public String savePiwoker() {              //新增巡检工基本信息
        this.getService().savePiwoker(piwoker);
        return "Piwokerlist";
    }
    public String preupdatePiwoker() {         //准备修改巡检工基本信息
        piwoker = this.getService().getOnePiwoker(piwoker.getWokerId());
        return "updatePiwoker";
    }
    public String updatePiwoker() {            //修改巡检工基本信息
        this.getService().updatePiwoker(piwoker);
        return "Piwokerlist";
    }
    public String deletePiwoker() {            //删除巡检工基本信息
        this.getService().deletePiwoker(piwoker.getWokerId());
        return "Piwokerlist";
    }
    public String findWokers() {               //脚本应用，到某个组下的巡检工
        String wokerStr = this.getService().forMenu(
                this.getPiwoker().getPiGroup().getGroupId());
        try {
            this.getResponse().setCharacterEncoding("UTF-8");
            this.getResponse().getWriter().println(wokerStr);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
    //省略属性 service、pageinfo、piwoker、pigroup 和 group 的 get()和 set()方法
...
}
```

接着在 struts.xml 文件中置关于模块操作的请求。

```
<!--配置名为 Piwokerlist 的 Action-->
<action name="Piwokerlist" method="Piwokerlist" class="piwoker-
Action">
   <result name="PiwokerlistSuccess">/xtgl/piwoker.jsp</result>
 </action>
<!--配置名为 presavePiwoker 的 Action-->
  <action name="presavePiwoker" method="presavePiwoker" class=
  "piwokerAction">
   <result name="success">/xtgl/piwokernew.jsp</result>
 </action>
<!--配置名为 savePiwoker 的 Action-->
 <action name="savePiwoker" method="savePiwoker" class="piwoker-
 Action">
   <result name="Piwokerlist" type="redirect">/xtgl/Piwokerlist.
   do</result>
 </action>
<!--配置名为 preupdatePiwoker 的 Action-->
```

```
 <action name="preupdatePiwoker" method="preupdatePiwoker" class=
 "piwokerAction">
   <result name="updatePiwoker">/xtgl/piwokerupdate.jsp</result>
 </action>
<!--配置名为 updatePiwoker 的 Action-->
 <action name="updatePiwoker" method="updatePiwoker" class=
 "piwokerAction">
   <result name="Piwokerlist" type="redirect">/xtgl/Piwokerlist.
   do</result>
 </action>
<!--配置名为 deletePiwoker 的 Action-->
 <action name="deletePiwoker" method="deletePiwoker" class=
 "piwokerAction">
   <result name="Piwokerlist" type="redirect">/xtgl/Piwokerlist.
   do</result>
 </action>
<!-- viewpiwoker -->
 <action name="viewpiwoker" method="viewpiwoker" class="piwoker-
 Ation">
   <result name="PiGrouplist" type="redirect">/xtgl/PiGrouplist.
   do</result>
 </action>
 <!-- viewpiwoker1 -->
 <action name="viewpiwoker1" method="viewpiwoker1" class="piwoker-
 Action">
   <result>/xtgl/viewpiwoker.jsp</result>
 </action>
<!--配置名为 findWokers 的 Action-->
 <action name="findWokers" method="findWokers" class="piwoker-
 Action">
   <result>/xtgl/eiditWork.jsp</result>
 </action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于巡检工功能涉及的页面有：实现显示巡检工的页面 piwoker.jsp，具体内容如代码 27.100 所示；实现增加巡检工的页面 piwokernew.jsp，具体内容如代码 27.101 所示；实现修改巡检工的页面 piwokerupdate.jsp，具体内容如代码 27.102 所示。

代码 27.100　显示巡检工页面：piwoker.jsp

```
...
    <table>
        <!--表格的相关标题-->
        <td >巡检工编号</td>
        <td >巡检工姓名</td>
        <td >电话 1</td>
        <td >电话 2</td>
        <td >操作</td></tr>
        <!--遍历结果-->
        <s:iterator value="pageinfo.pagedata">
            <td>${wokerId}</td>                          <!--输出巡检工编号-->
            <td>${workerName}</td>                       <!--输出巡检工姓名-->
            <td>${workerTel1}</td>                       <!--输出电话 1-->
            <td>${workerTel2}</td>                       <!--输出电话 2-->
            <!--实现修改功能-->
            <a href="./preupdatePiwoker.do?piwoker.wokerId=${wokerId}">
            <img src="../images/fix.gif" title="修改" border="0"></a>
```

```
            <!--实现删除功能-->
            <a href="./deletePiwoker.do?piwoker.wokerId=${wokerId}"
                onClick="return confirm('确定删除此信息吗? ')">
            <img src="../images/del.gif" title="删除" border="0"></a>
        </s:iterator>
        <!--关于分页功能-->
        <td>
            <font>共${pageinfo.allrecord}项</font>
            <font>每页${pageinfo.pagerecord}项</font>
            <font>当前第${pageinfo.curpage}页</font>
            <font>共${pageinfo.allpage}页</font>
            <s:url id="list"value="/xtgl/deptlist.do" includeParams
            ="false"></s:url>
            <s:if test="%{pageinfo.curpage!=1}">
            <a href="${list }?pageinfo.curpage=1">首页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.curpage-1 }">上
            一页</a>
            </s:if>
            <s:if test="%{pageinfo.curpage!=pageinfo.allpage}">
            <ahref="${list }?pageinfo.curpage=${pageinfo.curpage+1 }">下
            一页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.allpage }">尾页
            </a>
            </s:if>
            <input id="pagebox" size="1" name="pagebox"></input>
            <a onclick="goto('./deptlist.do?m=list&');" href="#">跳转</a>
            <input id="totalpage" type="hidden" value="${pageinfo.
            allpage }"
                name="totalpage"></input>
        </td>
    </table>
...
```

代码 27.101　添加巡检工页面：piwokernew.jsp

```
...
        <form action="../xtgl/savePiwoker.do" method="post">
            <table>
                <!--巡检工姓名输入框 -->
                <td align="center">巡检工姓名:
                <input name="piwoker.workerName">
                </td>
                <!--电话 1 输入框 -->
                <td align="center">电话 1:
                <input name="piwoker.workerTel1" id="tel" onblur="checkTel
                ('tel')">
                </td>
                <!--电话 2 输入框 -->
                <td align="center">电话 2:
                <input name="piwoker.workerTel2" id="te2" onblur="checkTel
                ('te2')">
                </td>
                <!--提交和返回按钮 -->
                <input type="submit" value="提交">
                <input type="button" value="返回" onclick="history.back
                ();">
            </table>
```

```
        </form>
...
```

代码 27.102　更新巡检工页面：piwokerupdate.jsp

```
...
        <form action="./updatePiwoker.do" method="post">
            <table >
                <!--巡检工编号输入框 -->
                <td align="center">巡检工编号:
                <input name="piwoker.wokerId" value="${piwoker.wokerId }
                "readonly="true">
                </td>
                <!--巡检工姓名输入框 -->
                <td align="center">巡检工姓名:
                <input name="piwoker.workerName" value="${piwoker.Worker-
                Name }">
                </td>
                <!--电话 1 输入框 -->
                <td align="center">电话 1:
                <input name="piwoker.workerTel1" value="${piwoker.Worker-
                Tel1 }">
                </td>
                <!--电话 2 输入框 -->
                <td align="center">电话 2:
                <input name="piwoker.workerTel2" value="${piwoker.worker-
                Tel2 }">
                </td>
                <!--提交和返回按钮 -->
                <input type="submit" value="提交">
                <input type="button" value="返回" onclick="history.back
                ();">
        </form>
...
```

8. 关于银行操作的表示层

关于银行操作的所有请求，都由 Struts 2.0 框架中名为 BankAction 的 Action 类来处理，该类的具体内容如代码 27.103 所示。

代码 27.103　关于银行操作的跳转：BankAction.java

```
...
public class BankAction extends BaseAction {
    //创建属性
    private BankService bankservice;
    private Pageinfo pageinfo = new Pageinfo();
    private Bank bank;
    public BankAction() {                                    //构造函数
    }
    public String list() {                                   //展现所有的银行网点
        Object wdid = this.getSession().getAttribute("wdid");
        String cond = "";
        if (wdid != null) {
            cond = " and A.id='" + wdid.toString() + "'";
        }
```

```
        pageinfo = this.getBankservice().getBanks(pageinfo.getCurpage(),
        cond);
        return "success";
    }
    public String save() {                              //新增银行网点
        this.getBankservice().saveBank(bank);
        return "list";
    }
    public String preupdate() {                         //准备修改
        bank = this.getBankservice().getBank(bank.getBankId());
        return "update";
    }
    public String update() {                            //修改银行网点
        this.getBankservice().updateBank(bank);
        return "list";
    }
    public String delete() {                            //修改银行网点
        this.getBankservice().deleteBank(bank.getBankId());
        return "list";
    }
    public String checkBankId() {                       //校验银行 ID
        String id = this.getRequest().getParameter("id");
        String rs = this.getBankservice().checkBankId(id);
        try {
            this.getResponse().getWriter().println(rs);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
    //省略属性 bankservice、pageinfo 和 bank 的 get()和 set()方法
...
}
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 banklist 的 Action-->
<action name="banklist" method="list" class="bankaction">
    <result name="success" type="dispatcher">
        /xtgl/bank.jsp
    </result>
</action>
<!--配置名为 savebank 的 Action-->
<action name="savebank" method="save" class="bankaction">
    <result name="list" type="redirect">
        /xtgl/banklist.do
    </result>
</action>
<!--配置名为 preupdate 的 Action-->
<action name="preupdate" method="preupdate"
    class="bankaction">
    <result name="update" type="dispatcher">
        /xtgl/bankupdate.jsp
    </result>
</action>
<!--配置名为 updatebank 的 Action-->
<action name="updatebank" method="update" class="bankaction">
    <result name="list" type="redirect">
        /xtgl/banklist.do
    </result>
```

```
</action>
<!--配置名为 deletebank 的 Action-->
<action name="deletebank" method="delete" class="bankaction">
    <result name="list" type="redirect">
        /xtgl/banklist.do
    </result>
</action>
<!--配置名为 checkBankId 的 Action-->
<action name="checkBankId" method="checkBankId"
    class="bankaction">
    <result>/xtgl/banknew.jsp</result>
</action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于银行网点功能涉及的页面有：实现显示银行网点的页面 bank.jsp，具体内容如代码 27.104 所示；实现增加银行网点的页面 banknew.jsp，具体内容如代码 27.105 所示；实现修改银行网点的页面 bankequpdate.jsp，具体内容如代码 27.106 所示。

代码 27.104　显示银行页面：bank.jsp

```
...
    <table>
        <!--表格的相关标题-->
        <td>银行 ID</td>
        <td >银行名称</td>
        <td >银行位置经度</td>
        <td >银行位置纬度</td>
        <td >银行 IP</td>
        <td>操作</td></tr>
        <!--遍历结果-->
        <s:iterator value="pageinfo.pagedata" id="bank">
            <td >${bankId}</td>                    <!--输出银行 ID -->
            <td >${bankName}</td>                  <!--输出银行名称-->
            <td >${bankLongitude}</td>             <!--输出银行位置经度-->
            <td >${bankLatitude}</td>              <!--输出银行位置纬度-->
            <td >${bankIp}</td>                    <!--输出银行 IP<-->
            <!--实现编辑功能-->
            <a href="./preupdate.do?bank.bankId=${bankId }">
            <img src="../images/fix.gif" title="编辑" border="0"></a>
            <s:if test="%{#bank.bankEquipments==null || #bank.bank-
            Equipments.size()==0}">
            <!--实现删除功能-->
            <a href="./deletebank.do?bank.bankId=${bankId }">
            <img src="../images/del.gif" title="删除" onClick="return
            confirm('确定删除此信息吗？')" border="0"></a>
            </s:if>
            <s:elseiftest="%{#bank.bankEquipments!=null&&#bank.bank-
            Equipments.size>0}">
            <!--实现查看设备明细功能-->
            <a href="./listbankEquipment.do?bank.bankId=${bankId }">
            <img src="../images/VIEW.GIF" title="查看设备明细" border=
            "0"></a>
            </s:elseif>
            <!--实现新增明细功能-->
            <a href="./presave.do?bankEquipment.bank.bankId=${bankId }">
```

```
            <img src="../images/ADD5.GIF" title="新增明细" border="0"></a>
            </s:iterator>
        <!--关于分页功能-->
        <td>
            <font>共${pageinfo.allrecord}项</font>
            <font>每页${pageinfo.pagerecord}项</font>
            <font>当前第${pageinfo.curpage}页</font>
            <font>共${pageinfo.allpage}页</font>
            <s:url id="list"value="/xtgl/deptlist.do" includeParams=
            "false"></s:url>
            <s:if test="%{pageinfo.curpage!=1}">
            <a href="${list }?pageinfo.curpage=1">首页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.curpage-1 }">上
            一页</a>
            </s:if>
            <s:if test="%{pageinfo.curpage!=pageinfo.allpage}">
            <ahref="${list }?pageinfo.curpage=${pageinfo.curpage+1 }">下
            一页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.allpage }">尾页
            </a>
            </s:if>
            <input id="pagebox" size="1" name="pagebox"></input>
            <a onclick="goto('./deptlist.do?m=list&');" href="#">跳转</a>
            <input id="totalpage" type="hidden" value="${pageinfo.
            allpage }"
                name="totalpage"></input>
        </td>
    </table>
...
```

代码 27.105　增加银行页面：banknew.jsp

```
...
    <form action="${save }" method="post">
        <table align="center" border="1" width="60%">
            <!--银行编号输入框 -->
            <td>银行编号
            <input type="text" name="bank.bankId"
                            onblur="checkBankId('bankId');" id="bankId">
             (不能使用汉字且长度小于 10 位)
            </td>
            <!--银行名称输入框 -->
            <td>银行名称
            <input type="text" name="bank.bankName">
            </td>
            <!--银行位置经度输入框 -->
            <td>银行位置经度
            <input type="text" name="bank.bankLongitude">
            </td>
            <!--银行位置纬度输入框 -->
            <td>银行位置纬度
            <input type="text" name="bank.bankLatitude">
            </td>
            <!--银行 IP 输入框 -->
            <td>银行 IP
            <input type="text" name="bank.bankIp">
```

```
            (Ip 格式:xxx.xxx.xxx.xxx)
            </td>
            <!--保存和取消按钮 -->
            <input type="submit" value="保存">
            <input type="button" value="取消" onclick="history.back();">
        </form>
...
```

代码 27.106　更新银行页面：bankequpdate.jsp

```
...
    <form action="${update }" method="post">
        <table >
            <!--设备流水 ID 输入框 -->
            <td >设备流水 ID
            <input type="text" name="bankEquipment.equipmenteachId"
                value="${bankEquipment.equipmenteachId }" readonly=
                "true">
            </td>
            <!--所属种类名输入框 -->
            <td >所属种类名
            <input type="hidden" name="bankEquipment.equipmenttype.
            equipmentId"
                value="${bankEquipment.equipmenttype.equipmentId}">
            <input type="text" name="bankEquipment.equipmenttype.
            equipmentName"
                value="${bankEquipment.equipmenttype.equipmentName }
                "readonly="true">
            </td>
            <!--所在银行输入框 -->
            <td >所在银行
            <input type="text" name="bankEquipment.bank.bankName"
                value="${bankEquipment.bank.bankName}" readonly="true">
            <input type="hidden" name="bankEquipment.bank.bankId"
                value="${bankEquipment.bank.bankId}">
            </td>
            <!--购入价值输入框 -->
            <td >购入价值
            <input type="text" name="bankEquipment.equipmentValue"
                value="${bankEquipment.equipmentValue }" readonly="true">
            </td>
            <!--购入时间输入框 -->
            <td >购入时间
            <input name="bankEquipment.equipmentBuydate"
                value="<s:date name="bankEquipment.equipmentBuydate"/>"
            </td>
            <!--设备状态输入框 -->
            <td >设备状态
            <s:set name="zt" value="#{0:'设备正常',1:'报检设备',2:'停用设备
            '}"></s:set>
            <s:select list="#request.zt" theme="simple"name="bank-
            Equipment.status">
            </s:select>
            </td>
            <!--设备折旧残值输入框 -->
            <td >设备折旧残值
            <input type="text" name="bankEquipment.depreciationValue"
```

```
                value="${ bankEquipment.depreciationValue}">
            </td>
            <!--提交和取消 -->
            <input type="submit" value="提交">
            <input type="button" value="取消" onclick="history.back();">
        </form>
...
```

9. 关于银行设备明细操作的表示层

关于银行设备明细操作的所有请求，都由 Struts 2.0 框架中名为 DepartmentAction 的 Action 类来处理，该类的具体内容如代码 27.107 所示。

代码 27.107　关于银行设备明细跳转：BankEquipmentAction.java

```
...
public class BankEquipmentAction extends BaseAction {
    //创建属性
    private BankEquipmentService bankEquipmentService;
    private BankEquipment bankEquipment;
    private Pageinfo pageinfo = new Pageinfo();
    private Bank bank;
    public BankEquipmentAction() {                  //构造函数
    }
    public String list() {                          //展现某个银行下的设备明细
        String cond = " and A.bank.id= '" + bank.getBankId() + "'";
        pageinfo = this.getBankEquipmentService().getBankEquipments(
                pageinfo.getCurpage(), cond);
        return "success";
    }
    public String presave() {                       //准备新增数据，准备两条数据
        bank = this.getBankEquipmentService().getBank(// 1. 准备银行网点
                bankEquipment.getBank().getBankId());
        // 2. 准备设备种类
        List equipmentTypes = this.getBankEquipmentService()
                .getAllEquipmentTypes();
        this.getRequest().setAttribute("equipmentTypes", equipment-
        Types);
        return "save";
    }
    public String save() {                          //新增银行设备明细
        this.getBankEquipmentService().saveBankEquipment(bankEquipment);
        return "list";
    }
    public String delete() {                        //删除一个银行设备明细
        this.getBankEquipmentService().deleteBankEquipment(
                bankEquipment.getEquipmenteachId());
        return "list";
    }
    public String preupdate() {                     //准备修改页面数据
        bankEquipment = this.getBankEquipmentService().getBankEquipment(
                bankEquipment.getEquipmenteachId());
        return "update";
    }
    public String update() {                        //修改银行设备明细

    this.getBankEquipmentService().updateBankEquipment(bankEquipment);
```

```
        return "list";
    }
    public String checkBankEquId() {            //验证银行设备
        String id = this.getRequest().getParameter("id");
        String rs = this.getBankEquipmentService().checkBankEquId(id);
        try {
            this.getResponse().getWriter().println(rs);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
    //省略属性 bankequipment、bankequipmentservice、pageinfo 和 bank 的 get()
    和 set()方法
...
}
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 listbankEquipment 的 Action-->
<action name="listbankEquipment" method="list"
    class="bankEquipmentaction">
    <result>/xtgl/bankEquipment.jsp</result>
</action>
<!--配置名为 presave 的 Action-->
<action name="presave" method="presave"
    class="bankEquipmentaction">
    <result name="save" type="dispatcher">
        /xtgl/bankequnew.jsp
    </result>
</action>
<!--配置名为 savebankEquipment 的 Action-->
<action name="savebankEquipment" method="save"
    class="bankEquipmentaction">
    <result name="list" type="redirect">
          /xtgl/listbankEquipment.do?bank.bankId=${bankEquipment.bank.
          bankId}
    </result>
</action>
<!--配置名为 deletebankEquipment 的 Action-->
<action name="deletebankEquipment" method="delete"
        class="bankEquipmentaction">
    <result name="list" type="redirect">
          /xtgl/listbankEquipment.do?bank.bankId=${bankEquipment.
          bank.bankId}
    </result>
</action>
<!--配置名为 preupdateEquipment 的 Action-->
<action name="preupdateEquipment" method="preupdate"
    class="bankEquipmentaction">
    <result name="update" type="dispatcher">
        /xtgl/bankequpdate.jsp
    </result>
</action>
<!--配置名为 updatebankEquipment 的 Action-->
<action name="updatebankEquipment" method="update"
    class="bankEquipmentaction">
    <result name="list" type="redirect">
/xtgl/listbankEquipment.do?bank.bankId=${bankEquipment.bank.bankId}
    </result>
```

```
</action>
<!--配置名为 checkBankEquId 的 Action-->
<action name="checkBankEquId" method="checkBankEquId"
    class="bankEquipmentaction">
    <result>/xtgl/bankequnew.jsp</result>
</action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于银行设备明细功能涉及的页面有：实现显示银行设备明细的页面 bankEquipment.jsp，具体内容如代码 27.108 所示；实现增加银行设备明细的页面 bankEquipmentnew.jsp，具体内容如代码 27.109 所示；实现修改银行设备明细的页面 bankequpdate.jsp，具体内容如代码 27.110 所示。

代码 27.108　显示银行设备明细页面：bankEquipment.jsp

```
...
    <table>
        <!--表格的相关标题-->
        <td>设备流水 ID</td>
        <td>银行编号 ID </td>
        <td>设备各类 ID </td>
        <td>购入价格</td>
        <td>购入时间</td>
        <td>设备状态</td>
        <td>设备折旧残值</td>
        <td>操作</td>
        <!--遍历结果-->
        <s:iterator value="pageinfo.pagedata">
            <td>${equipmenteachId }</td>           <!--输出设备流水 ID -->
            <td>${bank.bankName}</td>              <!--输出银行编号 ID -->
            <td>${equipmenttype.equipmentName}</td>
                                                   <!--输出设备各类 ID -->
            <td>${equipmentValue }</td>            <!--输出购入价格-->
            <!--输出购入时间-->
            <td><s:date name="%{equipmentBuydate}" format="yyyy-MM-dd"
            /></td>
            <!--设备状态-->
            <td>
                <s:if test="%{status==0}"> 设备正常</s:if>
                <s:elseif test="%{status==1}">报检设备</s:elseif>
                <s:else>停用设备</s:else>
            </td>
            <td>${depreciationValue}</td>          <!--输出设备折旧残值-->
            <td>
            <!--实现编辑功能-->
            <a href="./preupdateEquipment.do?bankEquipment.equipmenteach
                Id=${equipmenteachId }">
            <img src="../images/fix.gif" title="编辑" border="0"></a>
            <s:if test="faultRepairs.size()==0&&piEquipmentTables.size()
                ==0&&equipmentmaintains.size()==0">
            <!--实现删除功能-->
            <a href="./deletebankEquipment.do?bankEquipment.
            equipmenteachId=${equipmenteachId }&bankEquipment.bank.
            bankId=${bank.bankId }">
            <img src="../images/del.gif" title="删除"onClick="return
            confirm('确定删除此信息              吗？')" border="0"></a>
```

```
            </s:if>
        <!--关于分页功能-->
        <td>
            <font>共${pageinfo.allrecord}项</font>
            <font>每页${pageinfo.pagerecord}项</font>
            <font>当前第${pageinfo.curpage}页</font>
            <font>共${pageinfo.allpage}页</font>
            <s:url id="list"value="/xtgl/deptlist.do" includeParams
            ="false"></s:url>
            <s:if test="%{pageinfo.curpage!=1}">
            <a href="${list }?pageinfo.curpage=1">首页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.curpage-1 }">上
            一页</a>
            </s:if>
            <s:if test="%{pageinfo.curpage!=pageinfo.allpage}">
            <ahref="${list }?pageinfo.curpage=${pageinfo.curpage+1 }">下
            一页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.allpage }">尾页
            </a>
            </s:if>
            <input id="pagebox" size="1" name="pagebox"></input>
            <a onclick="goto('./deptlist.do?m=list&');" href="#">跳转</a>
            <input id="totalpage" type="hidden" value="${pageinfo.
            allpage }"
                name="totalpage"></input>
        </td>
    </table>
...
```

代码 27.109　增加银行明细设备页面：bankequnew.jsp

```
...
    <form action="${save }" method="post">
        <table>
            <!--设备流水 ID 输入框 -->
            <td>设备流水 ID
            <input type="text" name="bankEquipment.equipmenteachId" id=
            "bankEquId">
             (不能使用汉字且长度小于 10 位)
            </td>
            <!--所属种类名选择框 -->
            <td>所属种类名
            <s:select name="bankEquipment.equipmenttype.equipmentId"
                list="#request.equipmentTypes" listKey="equipmentId"
                listValue="equipmentName" theme="simple">
            </s:select>
            </td>
            <!--所在银行输入框 -->
            <td >所在银行
            <input type="text" name="bankEquipment.bank.bankName"
                value="${bankEquipment.bank.bankName}" readonly="true">
            <input type="hidden" name="bankEquipment.bank.bankId"
                value="${bankEquipment.bank.bankId}">
            </td>
            <!--购入价值输入框 -->
            <td >购入价值
            <input type="text" name="bankEquipment.equipmentValue"
                value="${bankEquipment.equipmentValue }" readonly="true">
```

```
            </td>
            <!--购入时间输入框 -->
            <td >购入时间
            <input name="bankEquipment.equipmentBuydate"
                value="<s:date name="bankEquipment.equipmentBuydate"/>"
            </td>
            <!--设备状态选择框 -->
            <td >设备状态
            <s:set name="zt" value="#{0:'设备正常',1:'报检设备',2:'停用设备
            '}"></s:set>
            <s:select list="#request.zt" theme="simple"name="bank-
            Equipment.status">
            </s:select>
            </td>
            <!--设备折旧残值输入框 -->
            <td >设备折旧残值
            <input type="text" name="bankEquipment.depreciationValue"
                value="${ bankEquipment.depreciationValue}">
            </td>
            <!--提交和取消按钮 -->
            <input type="submit" value="提交">
            <input type="button" value="取消" onclick="history.back();">
        </form>
...
```

代码 27.110　更新银行设备明细页面：bankequpdate.jsp

```
...
    <form action="${update }" method="post">
        <table >
            <!--设备流水输入框 -->
            <td >设备流水
            <input type="text" name="bankEquipment.equipmenteachId"
                value="${bankEquipment.equipmenteachId }" readonly="true">
            </td>
            <!--所属种类名输入框 -->
            <td >所属种类名
            <input type="hidden" name="bankEquipment.equipmenttype.
            equipmentId"
                value="${bankEquipment.equipmenttype.equipmentId}">
            <input type="text" name="bankEquipment.equipmenttype.
            equipmentName"
                value="${bankEquipment.equipmenttype.equipmentName }
                "readonly="true">
            </td>
            <!--所在银行输入框 -->
            <td >所在银行
            <input type="text" name="bankEquipment.bank.bankName"
                value="${bankEquipment.bank.bankName}" readonly="true">
            <input type="hidden" name="bankEquipment.bank.bankId"
                value="${bankEquipment.bank.bankId}">
            </td>
            <!--购入价值输入框 -->
            <td >购入价值
            <input type="text" name="bankEquipment.equipmentValue"
                value="${bankEquipment.equipmentValue }" readonly="true">
            </td>
```

```
            <!--购入时间输入框 -->
            <td >购入时间
            <input name="bankEquipment.equipmentBuydate"
                value="<s:date name="bankEquipment.equipmentBuydate" >"
            </td>
            <!--设备状态选择框 -->
            <td >设备状态
            <s:set name="zt" value="#{0:'设备正常',1:'报检设备',2:'停用设备
            '}"></s:set>
            <s:select list="#request.zt" theme="simple" name="bank-
            Equipment.status">
            </s:select>
            </td>
            <!--设备折旧残值输入框 -->
            <td >设备折旧残值
            <input type="text" name="bankEquipment.depreciationValue"
                            value="${ bankEquipment.depreciationValue}">
            </td>
            <!--提交和取消按钮 -->
            <input type="submit" value="提交">
            <input type="button" value="取消" onclick="history.back();">
            </p>
        </form>
...
```

10．关于银行设备类型操作的表示层

关于银行设备类型操作的所有请求，都由 Struts 2.0 框架中名为 EquipmentTypeAction 的 Action 类来处理，该类的具体内容如代码 27.111 所示。

代码 27.111　关于银行设备类型操作的跳转：EquipmentTypeAction.java

```
...
public class EquipmentTypeAction extends BaseAction {
    //创建属性
    private EquipmentTypeService service;
    private Pageinfo pageinfo = new Pageinfo();
    private List<Equipmenttype> EquipmentTypes;
    private Equipmenttype equipmenttype;
    public EquipmentTypeAction() {                    //构造函数
    }
    public String list() {                            //展现 EquipmentType
        if (pageinfo == null) {
            pageinfo.setCurpage(1);
        }
        pageinfo = this.getService()
                .getEquipmentType(pageinfo.getCurpage(), "");
        return "success";
    }
    public String save() {                            //新增功能
        this.getService().save(equipmenttype);
        return "save";
    }
    public String preupdate() {                       //准备修改数据
        equipmenttype = this.getService().getEquipmenttype(
                equipmenttype.getEquipmentId());
        return "preupdate";
```

```
    }
    public String update() {                          //修改功能
        this.getService().update(equipmenttype);
        return "update";
    }
    public String delete() {                          //删除功能
        this.getService().delete(equipmenttype.getEquipmentId().trim());
        return "delete";
    }
    public String checkEquipmentId() {                //银行设备种类 ID 是否存在
        String id = this.getRequest().getParameter("id");
        String rs = this.getService().checkEquipmentId(id);
        System.out.println("==" + rs + "**");
        try {
            this.getResponse().getWriter().println(rs);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
    public String checkEquipmentName() {          //银行设备种类名称是否存在
        String name = this.getRequest().getParameter("name");
        String rs = this.getService().checkEquipmentName(name);
        System.out.println("==" + rs + "**");
        try {
            this.getResponse().getWriter().println(rs);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
   //省略属性 equipmenttype、pageinfo、service 和 equipmenttypes 的 get()和 set()
    方法
...
}
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!-- 银行设备种类管理 -->
<!--分页展现数据 -->
<action name="equlist" method="list"
    class="equipmentTypeAction">
    <result name="success">/xtgl/equipmentType.jsp</result>
</action>
<!--新增数据  -->
<action name="equsave" method="save"
    class="equipmentTypeAction">
    <result name="save" type="redirect">
        /xtgl/equlist.do
    </result>
</action>
<!-- 准备修改数据 -->
<action name="equpreupdate" method="preupdate"
    class="equipmentTypeAction">
    <result name="preupdate">
        /xtgl/equipmentTypeupdate.jsp
    </result>
</action>
<!-- 修改数据 -->
<action name="equupdate" method="update"
```

```
    class="equipmentTypeAction">
    <result name="update" type="redirect">
        /xtgl/equlist.do
    </result>
</action>
<!--删除数据-->
<action name="equdelete" method="delete"
    class="equipmentTypeAction">
    <result name="delete" type="redirect">
        /xtgl/equlist.do
    </result>
</action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于银行设备类型功能涉及的页面有：实现显示银行设备类型的页面 equipmentType.jsp，具体内容如代码 27.112 所示；实现增加银行设备类型的页面 equipmentTypenew.jsp，具体内容如代码 27.113 所示；实现修改银行设备类型的页面 equipmentTypeupdate.jsp，具体内容如代码 27.114 所示。

代码 27.112　显示银行设备种类页面：equipmentType.jsp

```
...
    <table>
        <!--新增按钮 -->
        <td>
        <input name="button" type="image" value="新增" href='./equipment-
        Typenew.jsp'" />
        </td>
        <!--表格的相关标题-->
        <td>设备种类 ID</td>
        <td>设备种类名称</td>
        <td>操作</td>
        <!--遍历结果-->
        <s:iterator value="pageinfo.pagedata">
            <td>${equipmentId}</td>                <!--输出设备种类-->
            <td>${equipmentName}</td>              <!--输出种类名称-->
            <td>
            <!--实现编辑功能-->
                <a href="./equpreupdate.do?equipmenttype.equipmentId
                =${equipmentId}">
                <img src="../images/fix.gif" title="编辑" border="0"></a>
                <s:if test="bankEquipments.size()==0&&faultRepairs.size
                ()==0
                    &&piEquipmentTables.size()==0">
            <!--实现编辑功能-->
                <a href="./equdelete.do?equipmenttype.equipmentId=${
                equipmentId}">
                <img src="../images/del.gif" title="删除"
                    onClick="return confirm('确定删除此信息吗？')" ></a>
                </s:if>
            </td>
        <!--关于分页功能-->
        <td>
            <font>共${pageinfo.allrecord}项</font>
            <font>每页${pageinfo.pagerecord}项</font>
            <font>当前第${pageinfo.curpage}页</font>
            <font>共${pageinfo.allpage}页</font>
```

```
            <s:url id="list"value="/xtgl/deptlist.do" includeParams=
            "false"></s:url>
            <s:if test="%{pageinfo.curpage!=1}">
            <a href="${list }?pageinfo.curpage=1">首页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.curpage-1 }">上
            一页</a>
            </s:if>
            <s:if test="%{pageinfo.curpage!=pageinfo.allpage}">
            <ahref="${list }?pageinfo.curpage=${pageinfo.curpage+1 }">下
            一页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.allpage }">尾页
            </a>
            </s:if>
            <input id="pagebox" size="1" name="pagebox"></input>
            <a onclick="goto('./deptlist.do?m=list&');" href="#">跳转</a>
            <input id="totalpage" type="hidden" value="${pageinfo.
            allpage }"
                name="totalpage"></input>
        </td>
    </table>
...
```

代码 27.113　新增银行设备种类页面：equipmentTypenew.jsp

```
...
    <form action="${save }" method="post" onsubmit="return checkdata();">
            <!--设备种类输入框 -->
        <td>设备种类 ID
        <input type="text" name="equipmenttype.equipmentId"
            onblur="checkEquipmentId('EqumentId');" id="EqumentId">
        (不能使用汉字且长度小于 10 位)
        </td>
            <!--设备名称输入框 -->
        <td >设备名称
        <input type="text" name="equipmenttype.equipmentName"
            onchange="checkEquipmentName('EqumentName')" id="EqumentName">
        </td>
            <!--保存和返回按钮 -->
        <input type="submit" value="保存">
        <input type="button" value="返回" onclick="history.back();">
        </form>
...
```

代码 27.114　更新银行设备种类页面：equipmentTypeupdate.jsp

```
...
    <form action="${update }" method="post">
        <table align="center" border="1" width="60%">
            <!--设备种类输入框 -->
            <td>设备种类 ID
            <input type="text" name="equipmenttype.equipmentId"
                value="${equipmenttype.equipmentId}" readonly="true">
            </td>
            <!--设备名称输入框 -->
            <td>设备名称
            <input type="text" name="equipmenttype.equipmentName"
                value="${equipmenttype.equipmentName}">
```

```
            </td>
            <!--修改和返回按钮 -->
            <<input type="submit" value="修改">
            <input type="button" value="返回" onclick="history.back();">
        </form>
...
```

11. 关于银行报修设备类型操作的表示层

关于银行报修设备类型操作的所有请求，都由 Struts 2.0 框架中名为 DepartmentAction 的 Action 类来处理，该类的具体内容如代码 27.115 所示。

代码 27.115　操作表格 Fault_Repair_Type 的类：FaultRepairTypeAction.java

```
...
public class FaultRepairTypeAction extends BaseAction {
    //创建属性
    private FaultRepairTypeService service;
    private FaultRepairType faultRepairType;
    private List<FaultRepairType> faultRepairTypes;
    private Pageinfo pageinfo = new Pageinfo();
    public FaultRepairTypeAction() {                //构造函数
    }
    //展现所有的设备报修问题（为下面菜单做准备的数据，不需要分页,）
    public String list() {
        if (pageinfo == null) {
            pageinfo.setCurpage(1);
        }
        pageinfo = this.getService().getpageinfo(pageinfo.getCurpage(), "");
        return "success";
    }
    public String save() {                          //新增设备报修问题表对象
        this.getService().save(faultRepairType);
        return "view";
    }
    public String preUpdate() {                     //为修改页面做准备数据
        faultRepairType = this.getService().getOneFaultRepairType(
                faultRepairType.getPitypeId());
        System.out.println(faultRepairType.getPitypeId() + "-----"
                + faultRepairType.getPitypeValue());
        return "preupdate";
    }
    public String update() {                    //修改设备报修问题某一项记录
        this.service.update(faultRepairType);
        return "update";
    }
    public String delete() {                    //删除设备报修问题指定一项记录
        this.service.delete(faultRepairType.getPitypeId());
        return "delete";
    }
    public String checkFaultRepaitTypeName() { //校验问题类型名称是否存在
        String name = this.getRequest().getParameter("name");
        String rs = this.getService().checkFaultRepairtTypeName(name);
        try {
            this.getResponse().getWriter().println(rs);
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
        return null;
    }
    //省略属性 faultrepairtype、service、faultrepairtypes、pageinfo 的 set()
    和 get()的方法
...
}
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
<!--配置名为 frtlist 的 Action-->
<action name="frtlist" method="list"
    class="faultRepairTypeAction">
    <result name="success">/xtgl/faultRepairType.jsp</result>
</action>
<!--配置名为 frtsave 的 Action-->
<action name="frtsave" method="save"
    class="faultRepairTypeAction">
    <result name="view" type="redirect">
        /xtgl/frtlist.do
    </result>
</action>
<!--配置名为 frtpreUpdate 的 Action-->
<action name="frtpreUpdate" method="preUpdate"
    class="faultRepairTypeAction">
    <result name="preupdate">
        /xtgl/faultRepairTypeupdate.jsp
    </result>
</action>
<!--配置名为 frtupdate 的 Action-->
<action name="frtupdate" method="update"
    class="faultRepairTypeAction">
    <result name="update" type="redirect">
        /xtgl/frtlist.do
    </result>
</action>
<!--配置名为 frtdelete 的 Action-->
<action name="frtdelete" method="delete"
    class="faultRepairTypeAction">
    <result name="delete" type="redirect">
        /xtgl/frtlist.do
    </result>
</action>
<!--配置名为 checkFaultRepaitTypeName 的 Action-->
<action name="checkFaultRepaitTypeName"
    method="checkFaultRepaitTypeName" class="faultRepairTypeAction">
    <result>/xtgl/faultRepairType.jsp</result>
</action>
```

根据 struts.xml 文件中关于该类的配置信息，可以发现关于银行设备报修问题类型功能涉及的页面有：实现显示银行报修问题类型的页面 faultRepairType.jsp，具体内容如代码 27.116 所示；实现增加银行报修问题类型的页面 faultRepairTypenew.jsp，具体内容如代码 27.117 所示；实现修改银行报修问题类型的页面 faultRepairTypeupdate.jsp，具体内容如代码 27.118 所示。

代码 27.116　显示设备报修问题类型页面：faultRepairType.jsp

```
...
    <table>
        <!--新增按钮-->
        <td>
        <input name="button" type="image" src="../images/add.gif"
                class="input button9" value="新增" href='./faultRepair-
                Typenew.jsp'" /></td>
        <!--表格的相关标题-->
        <td>问题编号</td>
        <td>问题名称</td>
        <td>操作</td>
        <!--遍历结果-->
        <s:iterator value="pageinfo.pagedata">
            <td>${pitypeId}</td>                    <!--输出问题编号-->
            <td>${pitypeValue}</td>                 <!--输出问题名字-->
            <td>
            <!--实现编辑功能-->
            <a href="./frtpreUpdate.do?faultRepairType.pitypeId=${
            pitypeId}">
            <img src="../images/fix.gif" title="编辑" border="0"></a>
            <s:if test="piEquipmentTables.size()==0&&faultRepairs.size
            ()==0">
            <!--实现删除功能-->
            <a href="./frtdelete.do?faultRepairType.pitypeId=${
            pitypeId}">
            <img src="../images/del.gif" title="删除"
                onClick="return confirm('确定删除此信息吗？')" border="0">
                </a>
            </s:if>
            </td>
        </s:iterator>
        <!--关于分页功能-->
        <td>
            <font>共${pageinfo.allrecord}项</font>
            <font>每页${pageinfo.pagerecord}项</font>
            <font>当前第${pageinfo.curpage}页</font>
            <font>共${pageinfo.allpage}页</font>
            <s:url id="list"value="/xtgl/deptlist.do" includeParams=
            "false"></s:url>
            <s:if test="%{pageinfo.curpage!=1}">
            <a href="${list }?pageinfo.curpage=1">首页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.curpage-1 }">上
            一页</a>
            </s:if>
            <s:if test="%{pageinfo.curpage!=pageinfo.allpage}">
            <ahref="${list }?pageinfo.curpage=${pageinfo.curpage+1 }">下
            一页</a>
            <a href="${list }?pageinfo.curpage=${pageinfo.allpage }">尾页
            </a>
            </s:if>
            <input id="pagebox" size="1" name="pagebox"></input>
            <a onclick="goto('./deptlist.do?m=list&');" href="#">跳转</a>
            <input id="totalpage" type="hidden" value="${pageinfo.
```

```
            allpage }"
                name="totalpage"></input>
        </td>
    </table>
...
```

代码 27.117　增加设备报修问题页面：faultRepairTypenew.jsp

```
...
    <form action="${save }" method="post" onsubmit="return checkdata-
    RepairType();">
        <table>
            <!--问题类型名称输入框 -->
            <td >问题类型名称
            <input name="faultRepairType.pitypeValue"
                onchange="checkFaultRepaitTypeName('PitypeValue')" id=
                "PitypeValue">
            </td>
            <!--保存和返回按钮 -->
            <input type="submit" value="保存">
            <input type="button" value="返回" onclick="history.back();">
        </form>
...
```

代码 27.118　更新设备报修问题类型页面：faultRepairTypeupdate.jsp

```
...
    <form action="${update }" method="post">
        <table >
            <!--问题编号输入框-->
            <td>问题编号
            <input name="faultRepairType.pitypeId"
                value="${ faultRepairType.pitypeId}" readonly="readonly">
            </td>
            <!--问题名称输入框-->
            <td >问题名称
            <input name="faultRepairType.pitypeValue"
                value="${ faultRepairType.pitypeValue}">
            </td>
            <!--提交和返回按钮-->
            <input type="submit" value="提交">
            <input type="button" value="返回" onclick="history.back();">
        </form>
...
```

27.4　商业银行设备巡检系统具体实现——设备报修管理

为了让读者可以快速地理解和掌握商业银行设备巡检系统，在具体讲解时先按照面向应用的方式对该系统进行分模块：系统管理、设备巡检管理和设备报修管理，然后再对各个模块进行 MVC 分层。

本节将详细讲解设备报修管理应用，该模块按照 MVC 模式分成为领域模型层、持久层、业务层和表现层。本节将介绍关于系统管理的相关 4 层。

27.4.1　设备报修管理的领域模型层

在具体实现商业银行设备巡检系统中的设备报修管理模块时，主要涉及的表格有存放银行设备报修信息的表 Fault_Repair。

FaultRepair.java 类为表格 Fault_Repair 对应的领域模型对象类，该类的具体内容如代码 27.119 所示。该领域对象类的映射文件如代码 27.120 所示。

代码 27.119　表 Fault_Repair 领域模型对象：FaultRepair.java

```
...
public class FaultRepair  implements java.io.Serializable {
    //创建属性
    private Long repairid;                          //设备报修序列号
    private BankEquipment bankEquipment;            //银行设备明细表
    private Equipmenttype equipmenttype;            //设备种类表
    private Users users;                            //登录
    private Bank bank;                              //银行编码
    private PiGroup piGroup;                        //巡检组
    private FaultRepairType faultRepairType;        //设备报修问题类型表
    private Date faultRepairBeginDate;              //设备报修时间
    private String repairStatus;                    //报修记录状态
    private String allocateStatus;                  //小组分配状态
    private Date faultRepairEndDate;                //设备维修结束时间
    private String faultRepairEvaluation;           //设备维修情况描述
    public FaultRepair() {                          //无参构造函数
}
    //带参构造函数
    public FaultRepair(BankEquipment bankEquipment, Equipmenttype equip-
    menttype, Users users, Bank bank, PiGroup piGroup, FaultRepairType faul-
    tRepairType, Date faultRepairBeginDate, String repairStatus, String
    allocateStatus, Date faultRepairEndDate, String faultRepairEvaluation)
     {
        this.bankEquipment = bankEquipment;
        this.equipmenttype = equipmenttype;
        this.users = users;
        this.bank = bank;
        this.piGroup = piGroup;
        this.faultRepairType = faultRepairType;
        this.faultRepairBeginDate = faultRepairBeginDate;
        this.repairStatus = repairStatus;
        this.allocateStatus = allocateStatus;
        this.faultRepairEndDate = faultRepairEndDate;
        this.faultRepairEvaluation = faultRepairEvaluation;
    }
    //省略属性 repairid、bankequipment、equipmenttype、users、bank、pigroup、
    faultrepairtype、    faultrepairbeginDate、repairstatus、allocatestatus、
    faultrepairenddate 和 faultrepairevaluation  的 get()和 set()方法
...
}
```

代码 27.120　FaultRepair 类的映射文件：FaultRepair.hbm.xml

```
...
<hibernate-mapping>
    <!--制定要持久化类与对应数据库的表映射关系-->
    <class name="com.jb.syyh.po.FaultRepair" table="fault repair" schema=
    "public">
        <!--表 name 主键的设置-->
        <id name="repairid" type="java.lang.Long">
            <column name="repairid" />
            <generator class="sequence" />
        </id>
        <!--设置多对一关系-->
        <many-to-one name="bankEquipment" class="com.jb.syyh.po.Bank-
        Equipment"
                lazy="false" fetch="select">
            <column name="equipmenteach_id" length="10" />
        </many-to-one>
        <!--设置多对一关系-->
        <many-to-one name="equipmenttype" class="com.jb.syyh.po.Equipment-
        type"
                lazy="false" fetch="select">
            <column name="equipment_id" length="10" />
        </many-to-one>
        <!--设置多对一关系-->
        <many-to-one name="users" class="com.jb.syyh.po.Users" lazy="false"
        fetch="select">
            <column name="login id" length="10" />
        </many-to-one>
        <!--设置多对一关系-->
        <many-to-one name="bank" class="com.jb.syyh.po.Bank" lazy="false"
        fetch="select">
            <column name="bank_id" length="10" />
        </many-to-one>
        <!--设置多对一关系-->
        <many-to-one name="piGroup" class="com.jb.syyh.po.PiGroup"
                lazy="false" fetch="select">
            <column name="group id" />
        </many-to-one>
        <!--设置多对一关系-->
        <many-to-one name="faultRepairType" class="com.jb.syyh.po.Fault-
        epairType"
                lazy="false" fetch="select">
            <column name="pitype id" />
        </many-to-one>
        <!--表 FaultRepair 字段"fault_repair_begin_dat 与属性 faultRepair-
        eginDate 的映射关系-->
        <property name="faultRepairBeginDate" type="java.util.Date">
            <column name="fault repair begin date" length="29" />
        </property>
        <!--表 FaultRepair 字段"repair status 与属性 repairStatus 的映射关系-->
        <property name="repairStatus" type="java.lang.String">
            <column name="repair_status" length="1" />
        </property>
        <!--表 FaultRepair 字段 allocate_status 与属性 faultRepairBeginDate 的映
        射关系-->
        <property name="allocateStatus" type="java.lang.String">
            <column name="allocate status" length="1" />
        </property>
```

```
        <!--表FaultRepair字段fault repair end date与属性faultRepairEndDate
        的映射关系-->
        <property name="faultRepairEndDate" type="java.util.Date">
            <column name="fault repair end date" length="29" />
        </property>
        <!--表 FaultRepair 字段 fault_repair_evaluation 与属性 faultRepair-
        valuation 的映射关系-->
        <property name="faultRepairEvaluation" type="java.lang.String">
            <column name="fault repair evaluation" />
        </property>
    </class>
</hibernate-mapping>
```

注意：aultRepair.java 类可以参考数据库中的表格 Fault_Repair 来编写，而该类的映射文件则可以通过向导实现。

27.4.2　设备报修管理的持久层

持久层主要用来实现对每个数据库的表格进行各种操作，对于商业银行设备巡检系统中的设备报修管理模块的持久层，其实就是实现对各个领域模型对象的各种操作。涉及的类分别为操作银行设备报修信息的表 Fault_Repair 的接口类 FaultRepairDao，以及接口实现类 FaultRepairDaoi。

实现操作表 Fault_Repair 的接口为 FaultRepairDao.java，该类的具体内容如代码 27.121 所示；实现该接口的类 FaultRepairDaoi.java，该类的具体内容如代码 27.122 所示。

代码 27.121　操作表格 Fault_Repair 的接口：FaultRepairDao.java

```
...
public interface FaultRepairDao {
    List getFps(Integer curpage, Integer pagerecord, String cond);
                                              //根据分页条件查到报修记录
    Long getCount(String cond);               //根据条件得到总页数
    FaultRepair geyFp(Long id);               //根据编号得到一个记录
    List getlist(String cond);                //根据条件得到 list
    //关于设备报修记录操作
    void save(FaultRepair po);                //新增设备报修记录
    void update(FaultRepair po);              //修改设备报修记录
    void delete(FaultRepair po);              //删除设备报修记录
}
```

【代码解析】

- getFps()方法实现根据分页条件查到报修记录。
- getCount()方法用来获取报修记录的总记录数。
- geyFp()方法用来获取一个报修记录。
- getlist()方法用来获取符合条件的报修记录。
- save()方法用来保存报修记录。
- update()方法用来更新报修记录。
- delete()方法用来删除报修记录。

代码 27.122　操作表格 Fault_Repair 的类：FaultRepairDaoi.java

```
...
public class FaultRepairDaoi extends BaseDaoSupport implements Fault-
RepairDao {
    public FaultRepairDaoi() {                          //构造函数
    }
    public Long getCount(String cond) {                 //根据条件得到总页数
        String hql = "select count(A) from FaultRepair A where 1=1 " + cond;
        System.out.println(hql);
        return this.findValue(hql);
    }
    //根据条件查到报修记录
    public List getFps(Integer curpage, Integer pagerecord, String cond) {
        String hql = " from FaultRepair A where 1=1 " + cond
                + " order by  A.repairid desc";
        return this.find(hql, curpage, pagerecord);
    }
    public FaultRepair geyFp(Long id) {                 //根据编号得到一个记录
        return (FaultRepair) this.getHibernateTemplate().get(FaultRepair.
        class,id);
    }
    public List getlist(String cond) {                  //根据条件得到 list
        String hql = " from FaultRepair A where 1=1 " + cond
                + " order by  A.repairid desc";
        return this.getHibernateTemplate().find(hql);
    }

    //关于设备报修记录的操作
    public void save(FaultRepair po) {                  //新增设备报修记录
        this.getHibernateTemplate().save(po);
    }
    public void update(FaultRepair po) {                //修改设备报修记录
        this.getHibernateTemplate().update(po);
    }
    public void delete(FaultRepair po) {                //删除设备报修记录
        this.getHibernateTemplate().delete(po);
    }
}
```

注意：在上述代码中，分别实现了 FaultRepairDao 接口中的所有方法。

27.4.3　设备报修管理的业务层

业务层主要用来实现对每个数据库的表格进行各种逻辑操作，对于商业银行设备巡检系统中的设备报修管理模块的业务层，其实就是实现该系统的各种功能。涉及的类主要为实现设备报修各种功能的接口类 FaultRepairService 和接口实现类 FaultRepairServicei。

关于 FaultRepair 操作的接口为 FaultRepair.java，该类的具体内容如代码 27.123 所示。实现该接口的类为 FaultRepairServicei.java，该类的具体内容如代码 27.124 所示。

代码 27.123　实现关于 FaultRepair 业务的各种方法接口：FaultRepairService.java

```
...
public interface FaultRepairService {
    // 分页得到 pageinfo
    Pageinfo getPage(Integer curpage, Integer pagerecord, String cond);
    FaultRepair geyFp(Long id);                  //根据编号得到一个记录
    void save(FaultRepair po);                   //新增设备报修
    void update(FaultRepair po);                 //修改设备报修
    void delete(FaultRepair po);                 //删除设备报修
    //生成报表
    void scbb(String cond, HttpServletRequest request,
            HttpServletResponse response);
}
```

【代码解析】

- getPage()方法用来获取分页对象。
- geyFp()方法用来获取一个设备报修记录。
- save()方法用来实现保存设备报修记录。
- update()方法用来实现更新设备报修记录。
- delete()方法用来实现删除设备报修记录。
- scbb()方法用来实现生成报修。

代码 27.124　实现关于 FaultRepair 业务的各种方法类：FaultRepairServicei.java

```
...
public class FaultRepairServicei implements FaultRepairService {
    //创建属性
    private FaultRepairDao FaultRepair;
    public FaultRepairServicei() {                    //构造函数
    }
    public void delete(FaultRepair po) {              //删除设备报修
        this.FaultRepair.delete(po);
    }
    //分页得到 pageinfo
    public Pageinfo getPage(Integer curpage, Integer pagerecord, String
    cond) {
        int allrecord = (this.FaultRepair.getCount(cond)).intValue();
        List list = this.FaultRepair.getFps(curpage, pagerecord, cond);
        Pageinfo page = new Pageinfo(curpage, allrecord, pagerecord, list);
        return page;
    }
    public FaultRepair geyFp(Long id) {               //根据编号得到一个记录
        return this.FaultRepair.geyFp(id);
    }
    public void save(FaultRepair po) {                //新增设备报修
        po.setRepairid(null);
        this.FaultRepair.save(po);
    }
    public void update(FaultRepair po) {              //修改设备报修
        this.FaultRepair.update(po);
    }
    //生成报表
    public void scbb(String cond, HttpServletRequest request,
```

```
            HttpServletResponse response) {
...
    }
    //省略属性 faultrepair 的 set()和 get()方法
    ...
}
```

注意：在上述代码中，分别实现了 FaultRepairService 接口中的所有方法。

27.4.4　设备报修管理的表示层

在设计实现关于商业银行设备巡检系统中的设备报修管理的表现层时，利用 Struts 2.0 框架来实现页面的跳转。涉及的类为实现设备报修操作页面跳转的类 FaultRepairAction。

关于设备报修操作的所有请求，都由 Struts 2.0 框架中名为 FaultRepairAction 的 Action 类来处理，该类的具体内容如代码 27.125 所示。

代码 27.125　关于设备报修页面操作的跳转：FaultRepairAction.java

```
...
public class FaultRepairAction extends BaseAction {
    private BankEquipmentService bankEquipment;         //银行设备明细
    private EquipmentTypeService EquipmentType;         //设备种类
    private PiGroupService piGroup;                     //巡检组
    private FaultRepairService FaultRepair;             //设备报修明细
    private FaultRepairTypeService faultRepairType;     //问题列表
    private SbwxService sbwx;                           //维修记录
    private BankService bank;                           //银行网点
    private FaultRepair po;                             //设备报修 po
    private Pageinfo pageinfo;                          //分页 page
    private String rs;                                  //报修记录状态
    private String as;                                  //小组分配状态
    public FaultRepairAction() {                        //构造函数
    }
    public String listxj() {            //得到前 4 个未分配小组的巡检记录
        String cond = " and A.repairStatus='0' and A.allocateStatus='0'   ";
        pageinfo = this.FaultRepair.getPage(1, 4, cond);
        return "listxj";
    }
    public String list() {              //巡检中心根据条件分页得到巡检记录
    }
    public String listg4() {            //得到工作组在某个网点要维修的前 4 条记录
    }
    public String listg() {             //分页得到工作组在某个网点要维修的所有记录
    }
    public String listw4() {            //网点报修需确认的前 4 条记录
    }
    //分页得到网点报修的所有记录（初始条件是需确认的）
    public String listw() {
    }
    public String presavew() {          //银行网点准备新增
    }
```

```
    public String presavez() {                  //巡检中心准备新增
    }
    public String getsbmx() {                   //得到设备明细
    }
    public String savez() {                     //巡检中心新增设备巡检
    }
    public String savew() {                     //网点新增设备巡检
    }
    public String wtlxsave() {                  //维修工新增问题类型
    }
    public String preupdatew() {                //网点提交前准备修改
    }
    public String preupdateg1() {               //维修工确认准备修改 1
    }
    public String preupdateg2() {               //维修工确认准备修改 2
    }
    public String preupdatez() {                //巡检中心提交前准备修改
    }
    public String preupdatezq1() {              //巡检中心分配小组准备修改 1
    }
    public String preupdatezq2() {              //巡检中心分配小组准备修改 2
    }
    public String updatew() {                   //网点提交前修改
    }
    public String updatez() {                   //巡检中心提交前修改
    }
    public String updatewt() {                  //网点提交修改
    }
    public String updatezt() {                  //巡检中心提交修改
    }
    public String updatezq1() {                 //巡检中心分配小组修改 1
    }
    public String updatezq2() {                 //巡检中心分配小组修改 2
    }
    public String updateg1() {                  //维修工确认修改 1
    }
    public String updateg2() {                  //维修工确认修改 2
    }
    public String updatewq1() {                 //网点确认修改 1
    }
    public String updatewq2() {                 //网点确认修改 2
    }
    public String deletew() {                   //网点未提交前删除
    }
    public String deletez() {                   //维修中心未提交前删除
    }
    public String getfr() {                     //实现查看功能
    }
    public String getwtlx() {                   //模糊查询
    }
    public String scbb() {                      //导出报表
    }
    //省略属性 faultrepair、po、pageinfo、bankequipment、equipmenttype、bank、
    pigroup、as、rs、faultrepairtype 和 sbwx 的 get()和 set()方法
    ...
```

```
}
```

接着在 struts.xml 文件中配置关于模块操作的请求。

```
...
<struts>
    <package name="sbbx" namespace="/sbbx" extends="AccessSYYH">
     <action name="sbbx!*" method="[1]" class="sbbxAction">
       <result name="listxj">../sbbx/xjlogin.jsp</result>
                                        <!--巡检中心登录初始界面-->
       <result name="list">../sbbx/xjlist.jsp</result>
                                        <!--巡检中心分页查询-->
       <result name="listg4">../sbbx/glogin.jsp</result>
                                        <!--维修工登录初始界面-->
       <result name="listg">../sbbx/glist.jsp</result>
                                        <!--维修工分页查询-->
       <result name="listw4">../sbbx/wdlogin.jsp</result>
                                        <!--网点登录初始界面-->
       <result name="listw">../sbbx/wdlist.jsp</result>
                                        <!--网点分页查询-->
       <result name="presavew">../sbbx/wdsave.jsp</result>
                                        <!--网点新增页面-->
       <result name="presavez">../sbbx/xjsave.jsp</result>
                                        <!--巡检中心新增页面-->
       <!--巡检中心新增完成转向-->
       <result name="savez" type="redirect">../sbbx/sbbx!list.do</result>
       <!--网点新增完成转向-->
       <result name="savew" type="redirect">../sbbx/sbbx!listw.do</
       result>
       <!--网点提交前准备修改转向-->
       <result name="preupdatew">../sbbx/wdupdate.jsp</result>
       <!--巡检中心提交前准备修改转向-->
        <result name="preupdatez">../sbbx/xjupdate.jsp</result>
       <result name="preupdateg1">../sbbx/gwx1.jsp</result><!--维修工准备
       维修转向 1-->
       <result name="preupdateg2">../sbbx/gwx2.jsp</result><!--维修工准备
       维修转向 2-->
       <!--巡检中心准备分配小组转向 1-->
       <result name="preupdatezq1">../sbbx/xjfpxz1.jsp</result>
       <!--巡检中心准备分配小组转向 2-->
       <result name="preupdatezq2">../sbbx/xjfpxz2.jsp</result>
       <!--网点提交转向-->
       <result name="updatewt" type="redirect">
             ../sbbx/sbbx!listw.do?pageinfo.curpage=${pageinfo.curpage}
             </result>
       <!--巡检中心提交转向-->
       <result name="updatezt" type="redirect">
             ../sbbx/sbbx!list.do?pageinfo.curpage=${pageinfo.curpage}
                                                         </result>
       <!--网点提交前修改转向-->
       <result name="updatew" type="redirect">
             ../sbbx/sbbx!listw.do?pageinfo.curpage=${pageinfo.curpage}
             </result>
       <!--巡检中心提交前修改转向-->
       <result name="updatez" type="redirect">
```

```
            ../sbbx/sbbx!list.do?pageinfo.curpage=${pageinfo.curpage}<
            /result>
        <!--巡检中心分配小组转向 1-->
        <result name="updatezq1" type="redirect">../sbbx/sbbx!listxj.do
        </result>
        <!--巡检中心分配小组转向 2-->
        <result name="updatezq2" type="redirect">
            ../sbbx/sbbx!list.do?pageinfo.curpage=${pageinfo.curpage}<
            /result>
        <!--维修工维修转向 1-->
        <result name="updateg1" type="redirect">../sbbx/sbbx!listg4.do</
        result>
        <!--维修工维修转向 2-->
        <result name="updateg2" type="redirect">
            ../sbbx/sbbx!listg.do?pageinfo.curpage=${pageinfo.curpage}
            </result>
        <!--网点确认维修转向 1-->
        <result name="updatewq1" type="redirect">../sbbx/sbbx!listw4.do
        </result>
        <!--网点确认维修转向 1-->
        <result name="updatewq2" type="redirect">
            ../sbbx/sbbx!listw.do?pageinfo.curpage=${pageinfo.curpage}
            </result>
        <!--网点提交前删除转向-->
        <result name="deletew" type="redirect">
            ../sbbx/sbbx!listw.do?pageinfo.curpage=${pageinfo.curpage}
            </result>
        <!--巡检中心提交前删除转向-->
        <result name="deletez"  type="redirect">
            ../sbbx/sbbx!list.do?pageinfo.curpage=${pageinfo.curpage}<
            /result>
        <result name="fr">../sbbx/sbbxview.jsp</result><!--查看设备报修转
        向-->

      </action>
    </package>
</struts>
```

由于篇幅的原因，所以本节将不具体介绍实现设备报修管理的具体页面，感兴趣的读者可以查看具体代码的 sbbx 文件，该文件夹内容如图 27.53 所示。

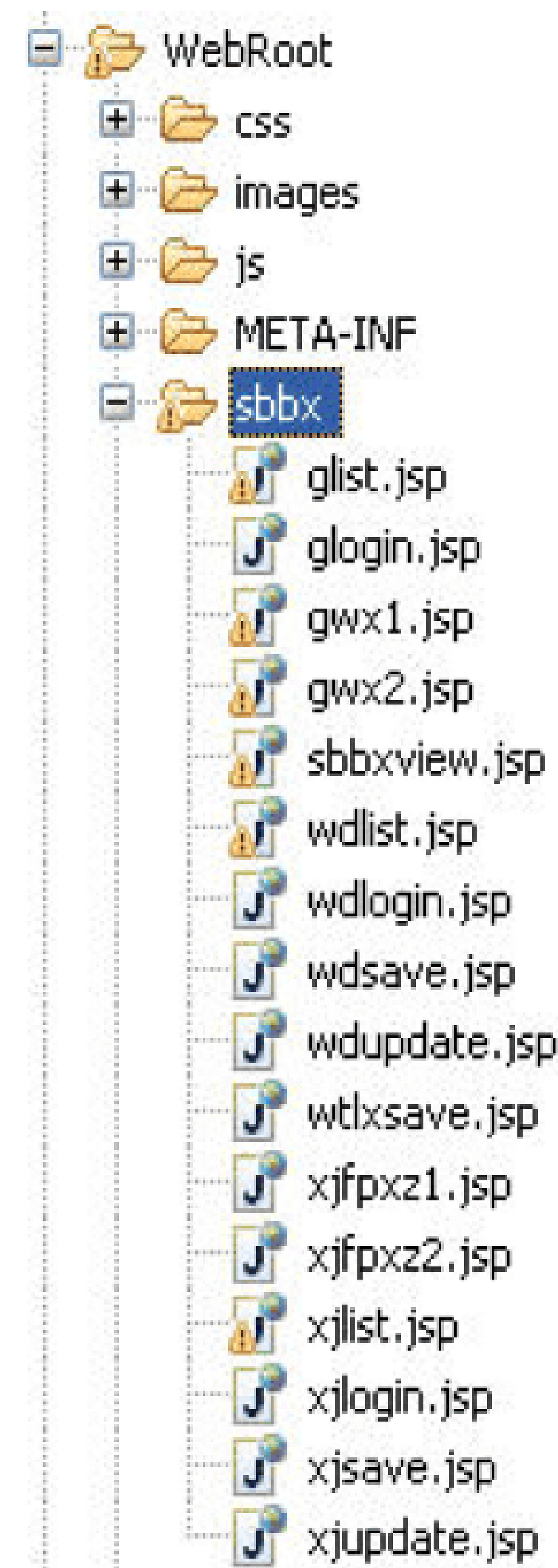


图 27.53　关于设备报修页面

27.5　商业银行设备巡检系统具体实现——设备巡检管理

为了让读者可以快速地理解和掌握商业银行设备巡检系统，在具体讲解时先按照面向应用的方式对该系统进行分模块：系统管理、设备巡检管理和设备报修管理，然后再对各个模块进行 MVC 分层。

本节将详细讲解设备巡检管理应用，该模块按照 MVC

模式分成为领域模型层、持久层、业务层和表现层。本节将介绍关于设备巡检管理的后三层，由于领域模型层是建立在其他领域模型的基础上，所以就不需要创建。

27.5.1　设备巡检管理的持久层

持久层主要用来实现对每个数据库表格进行各种操作，对于商业银行设备巡检系统中的设备巡检管理模块的持久层，其实就是实现对各个领域模型对象的各种操作。涉及的类分别为添加维修记录的接口类 SbwxDao 和接口实现类 SbwxDaoi，以及实现设备巡检的接口类 SbxjDao 和接口实现类 SbxjDaoi。

1. 添加维修记录的相关接口和类

实现添加维修记录的接口为 SbwxDao.java，该类的具体内容如代码 27.126 所示。实现该接口的类为 SbwxDaoi.java，该类的具体内容如代码 27.127 所示。

代码 27.126　添加维修记录的接口：SbwxDao.java

```
...
public interface SbwxDao {
    void saveEquipmentmaintain(Equipmentmaintain sbwx); //新增维修明细记录
    String getSbwxId(String cond);                      //获取维修明细记录
}
```

【代码解析】

- ❑ saveEquipmentmaintain()方法用来实现添加维修明细对象。
- ❑ getSbwxId()方法用来获取维修明细对象。

代码 27.127　添加维修记录的类：SbwxDaoi.java

```
...
public class SbwxDaoi extends BaseDaoSupport implements SbwxDao {
    public void saveEquipmentmaintain(Equipmentmaintain sbwx) {
                                                        //新增维修记录
        this.getHibernateTemplate().save(sbwx);
    }
    public String getSbwxId(String cond) {          //获取维修明细记录
        String hql = "select max(A.id) from Equipmentmaintain A where 1=1"
                + cond;
        Object obj = this.executeHQLForObject(hql, null);
        System.out.println(obj == null ? 0 : obj.toString());
        return obj == null ? "1" : Integer.parseInt(obj.toString()) + 1 +
        "";
    }
}
```

注意：在上述代码中，分别实现了 SbwxDao 接口中的所有方法。

2. 操作巡检记录的相关接口和类

实现操作巡检记录的接口为 SbxjDao.java，该类的具体内容如代码 27.128 所示。实现

该接口的类为 SbxjDaoi.java，该类的具体内容如代码 27.129 所示。

代码 27.128　实现设备巡检的接口：SbxjDao.java

```
...
public interface SbxjDao {
    Long count(String cond, Object... params);          //总记录数
    List getXbxjs(int curpage, int pagerecord, String cond, Object...
    params);                                            //某一页的记录
    void saveSbxj(PiEquipmentTable sbxj);               //新增巡检记录
    void updateSbxj(PiEquipmentTable sbxj);             //更新巡检记录;
    PiEquipmentTable getSbxj(Long id);                  //得到某个巡检记录
}
```

【代码解析】

- ❑ count()方法用来获取总记录数。
- ❑ getXbxjs()方法用来获取一页记录。
- ❑ saveSbxj()方法用来实现保存巡检记录对象。
- ❑ updateSbxj()方法用来实现更新巡检记录对象。
- ❑ getSbxj()方法用来获取一个巡检记录对象。

代码 27.129　实现设备巡检的类：SbxjDaoi.java

```
...
public class SbxjDaoi extends BaseDaoSupport implements SbxjDao {
    public SbxjDaoi() {                                        //构造函数
    }
    //为实现分页功能做准备
    public Long count(String cond, Object... params) {         //获取总记录数
        String hql = "select count(A) from PiEquipmentTable A where 1=1" +
        cond;
        return (Long) this.getDataCountWithHQL(hql, params);
    }
    public List getXbxjs(int curpage, int pagerecord, String cond,
                                                               //获取某一页数据
            Object... params) {
        String hql = "from PiEquipmentTable A  where 1=1" + cond
                + " order by A.id desc";
        return this.getPaginationDataWithHQL(hql, curpage, pagerecord,
        params);
    }
    //关于巡检记录的操作
    public void saveSbxj(PiEquipmentTable sbxj) {           //新增巡检记录
        this.getHibernateTemplate().save(sbxj);
    }
    public void updateSbxj(PiEquipmentTable sbxj) {         //更新巡检记录
        this.getHibernateTemplate().update(sbxj);
    }
    public PiEquipmentTable getSbxj(Long id) {              //获取某个巡检记录
        String hql = "from PiEquipmentTable A where A.id = " + id;
        return (PiEquipmentTable) this.executeHQLForObject(hql);
    }
}
```

注意：在上述代码中，分别实现了 SbxjDao 接口中的所有方法。

27.5.2　设备巡检管理的业务层

业务层主要用来实现对每个数据库表格进行各种逻辑操作，对于商业银行设备巡检系统中的设备巡检管理模块的业务层，其实就是实现该系统的各种功能。涉及的类分别为实现关于维修记录各种功能的接口类 SbwxService 和接口实现类 SbwxServicei，以及实现关于设备巡检各种功能的接口类 SbxjService 和接口实现类 SbxjServicei。

1．关于SbwxService业务方面的接口和类

关于 Sbwx 操作的接口为 SbwxService.java，该类的具体内容如代码 27.130 所示。实现该接口的类为 SbwxServicei.java，该类的具体内容如代码 27.131 所示。

代码 27.130　关于 Sbwx 业务方面的接口：SbwxService.java

```
...
public interface SbwxService {
    void saveEquipmentmaintain(Equipmentmaintain sbwx); // 新增维修记录
}
```

注意：上述代码中的 saveEquipmentmaintain()方法用来实现添加维修记录。

代码 27.131　关于 Sbwx 业务方面的类：SbwxServicei.java

```
...
public class SbwxServicei implements SbwxService {
    private SbwxDao dao;                                    //创建属性 dao
    public SbwxServicei() {                                 //构造函数
    }
    //实现保存功能
    public void saveEquipmentmaintain(Equipmentmaintain sbwx) {
        this.getDao().saveEquipmentmaintain(sbwx);
    }
    //省略属性 dao 的 set()和 get()方法
    ...
}
```

注意：在上述代码中，分别实现了 SbwxService 接口中的所有方法。

2．关于Sbxj业务方面的接口和类

关于 Sbwx 操作的接口为 SbwxService.java，该类的具体内容如代码 27.132 所示。实现该接口的类为 SbwxServicei.java，该类的具体内容如代码 27.133 所示。

代码 27.132　实现关于 Sbxj 业务的各种方法接口：SbxjService.java

```
...
public interface SbxjService {
    int pagerecord = 10;                                        //分页单位
```

```
    Pageinfo getSbxjs(int curpage, String cond,Object...params);
                                                          //分页数据
    void save(PiEquipmentTable sbxj);                     //巡检人员确认
    void updateSbxj(String logId,PiEquipmentTable sbxj);  //银行确认
    PiEquipmentTable getSbxj(Long sbId);            //得到某个设备巡检对象
}
```

【代码解析】

- getSbxjs()方法用来获取分页对象。
- save()方法用来实现巡检工确认。
- updateSbxj()方法用来实现银行确认。
- getSbxj()方法用来获取一个设备巡检对象。

代码 27.133　实现关于 Sbxj 业务的各种方法类：SbxjServicei.java

```
...
public class SbxjServicei implements SbxjService {
    private SbxjDao dao;                                //设备巡检 dao
    private BankEquipmentDao sbmxDaoi;                  //银行设备明细 dao
    private SbwxDao sbwxDaoi;
    //获取设备巡检记录中关于银行设备的记录
    public List<BankEquipment> getSbmxes(String cond, String bankId,
            Object... params) {
        List sbmxes = this.getSbmxDaoi().getBankEquipments(bankId, "ss");
                                                        //所有设备明细
        return sbmxes;
    }
    //执行查询得到的分页数据
    public Pageinfo getSbxjs(int curpage, String cond, Object... params) {
        Long allrecord = this.getDao().count(cond, params);
        List pagedata = this.getDao().getXbxjs(curpage, pagerecord, cond,
                params);
        Pageinfo pageinfo = new Pageinfo(curpage, allrecord.intValue(),
                pagerecord, pagedata);
        return pageinfo;
    }
    //巡检工确认
    public void save(PiEquipmentTable sbxj) {
        try {
            //将巡检表中的巡检确认设置为结束巡检
            sbxj.setPiStatus("0");
            sbxj.setPiDate(new Date());             //巡检时间
            if ("1".equals(sbxj.getStatus())) {     //如果设备的状态是异常的
                this.getDao().saveSbxj(sbxj);
                Equipmentmaintain sbwx = new Equipmentmaintain();
                                                    //创建设备维修对象
                BankEquipment bEquipment = new BankEquipment(); //银行设备
                bEquipment.setEquipmenteachId(sbxj.getBankEquipment()
                        .getEquipmenteachId());
                sbwx.setBankEquipment(bEquipment);
                sbwx.setMaintainDate(new Date());
                sbwx.setMaintainResult(sbxj.getPiEvaluation());
                this.getSbwxDaoi().saveEquipmentmaintain(sbwx);
            } else if ("0".equals(sbxj.getStatus())) {
                PiEquipmentTable po = new PiEquipmentTable();
                po.setBank(sbxj.getBank());
```

```
                po.setPiDate(new Date());
                po.setPiGroup(sbxj.getPiGroup());
                po.setPiStatus("0");
                po.setStatus("0");
                this.getDao().saveSbxj(po);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    //银行确认
    public void updateSbxj(String logId, PiEquipmentTable sbxj) {
        if (sbxj != null && logId != null && logId != "") {
            sbxj.setPiStatus("1");  //更改巡检确认为 "银行确认结束巡检"
            Users user = new Users();
            user.setLoginId(logId);
            sbxj.setUsers(user);    //将网点人员的 ID 作为巡检记录中的登录 ID
            this.getDao().updateSbxj(sbxj);
        }
    }
    public PiEquipmentTable getSbxj(Long sbId) {   //得到某个设备巡检对象
        return this.getDao().getSbxj(sbId);
    }
    //省略属性 dao、sbmxdaoi、sbwxdaoi 的 get()和 set()方法
    ...
}
```

注意：在上述代码中，分别实现了 SbxjService 接口中的所有方法。

27.5.3　设备巡检管理的表示层

在设计实现关于商业银行设备巡检系统中设备巡检管理的表现层时，利用 Struts 2.0 框架来实现页面的跳转。涉及的类为实现设备巡检操作页面跳转的类 SbxjActionAction。

关于设备巡检操作的所有请求都由 Struts 2.0 框架中名为 SbxjAction 的 Action 类来处理，该类的具体内容如代码 27.134 所示。

代码 27.134　关于设备巡检页面的跳转：SbxjAction.java

```
...
public class SbxjAction extends BaseAction {
    //创建各种字段
    private SbxjService service;                    //设备巡检 Service
    private BankServicei bankServicei;              //银行 Service
    private EquipmentTypeService sblxServicei;      //种类 Service
    private FaultRepairTypeService quesTypeServicei
                                                    //设备问题类型 Service
    private BankEquipmentService sbmxServicei;      //银行设备明细 Service
    private PiGroupService groupServicei;
    private Pageinfo pageinfo;
    private PiEquipmentTable sbxj;
    private String y;
    public SbxjAction() {                           //构造函数
    }
```

```
    //省略属性 y 的 set()和 get()方法
    ...
    public String preSave() {                          //为新增功能准备数据
        //获取银行 ID
        String bankId = (String) this.getSession().getAttribute("wdid");
        //获取巡检组 ID
        Long groupId = (Long) this.getSession().getAttribute("gid");
        if (bankId != null && bankId != "" && groupId != null) {
                                                       //判断银行 ID 和巡检组 ID
        //准备数据
            List quesList = this.getQuesTypeServicei().getAllFault-
            RepairTypes();
            this.getRequest().setAttribute("quesList", quesList);
            List sblxes = this.getSblxServicei().getAllEquipmentTypes();
                                                       //设备种类
            this.getRequest().setAttribute("sblxes", sblxes);
            return "save";
        } else {
            return "nosave";
        }
    }
    public String getSblsmxes() {
        //获取银行 ID
        String bankId = (String) this.getSession().getAttribute("wdid");
        //获取设备 ID
        String equipmentId = sbxj.getEquipmenttype().getEquipmentId();
        List list = this.getSbmxServicei().getBankEquipments(bankId,
                equipmentId);                      //获取设备报修记录
        String sblsmx = list.toString().substring(1,
                list.toString().length() - 1);
        this.getResponse().setCharacterEncoding("UTF-8");
        try {
            PrintWriter out = this.getResponse().getWriter();
            out.print(sblsmx);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
    //巡检人员巡检（巡检记录，维修记录）并进行确认
    public String save() {
        this.getService().save(sbxj);
        return "success";
    }
    //网点进行确认
    public String wdryList() {                 //准备数据(状态为结束巡检的记录)
        String bankId = (String) this.getSession().getAttribute("wdid");
        String cond = " and A.bank.bankId = '" + bankId + "'";
        int pageno = 1;
        if (pageinfo != null) {
            if (pageinfo.getCurpage() != 0) {
                pageno = pageinfo.getCurpage();
            }
        }
        pageinfo = this.getService().getSbxjs(pageno, cond, null);
        return "wdList";
```

```
    }
    public String updateSbxj() {            //执行网点确认
        Users user = (Users) this.getSession().getAttribute("user");
                                            //得到网点工作人员
        if (user != null) {
            String logId = user.getLoginId();
            //通过 ID 得到一个巡检对象
            sbxj = this.getService().getSbxj(sbxj.getId());
        this.getService().updateSbxj(logId, sbxj);
            return this.wdryList();
        }
        return null;
    }
    public String query() {                 //根据条件查询
        String startDate = this.getRequest().getParameter("date1");
        String endDate = this.getRequest().getParameter("date2");
        String wdmc = this.getRequest().getParameter("wdmc");
        //取查询条件输入框中的巡检组 Id
...
    }
    public String list() {                  //无条件查询
...
    }
    public String queryOne() {              //查看某条巡检记录
        sbxj = this.getService().getSbxj(sbxj.getId());
        return "showSbxj";
    }
    //省略属性 pageinfo、service、sbxj、typeservicei、
    sblxservicei、 bankService、sbmxServicei、
    groupservicei 的 set()和 get()方法
...
}
```

由于篇幅所限，本节将不具体介绍实现设备巡检管理的具体页面，感兴趣的读者可以查看具体代码的 sbxj 文件，该文件夹的内容如图 27.54 所示。



图 27.54　关于设备报修页面

27.6　多学两招——关于 PostgreSQL 数据库

每当谈论起计算机操作系统，肯定会涉及自由软件操作系统 Linux。可是如果说起数据库管理系统，可能绝大多数人只会记得有 Oracle、IBM DB2、Informix、Sybase、MS SQL Server，以及在互联网广为使用的轻量级数据库管理系统 MySQL。那么什么是 PostgreSQL 数据库管理系统？下面将会详细介绍。

27.6.1　下载 PostgreSQL 数据库管理系统

PostgreSQL 数据库管理系统起源于大名鼎鼎的“加州大学伯克利分校（BSD）”中的 Ingres 项目，该数据库管理系统不仅是全功能的自由软件数据库，而且还支持事务、子查

询和数据完整性检查。PostgreSQL 数据库管理系统具体下载步骤如下。

（1）首先访问下载 PostgreSQL 数据库的官方网站（http://www.postgresql.org/），如图 27.55 所示。在该页面中单击 Download 链接就可以转到与该软件相关的下载页面。



图 27.55　Postgresql 数据库首页

（2）在 PostgreSQL 数据库相关下载页面中（如图 27.56 所示），单击 pgInstaller 目录下的 Download 链接就可以转到关于该软件下载的真正页面。

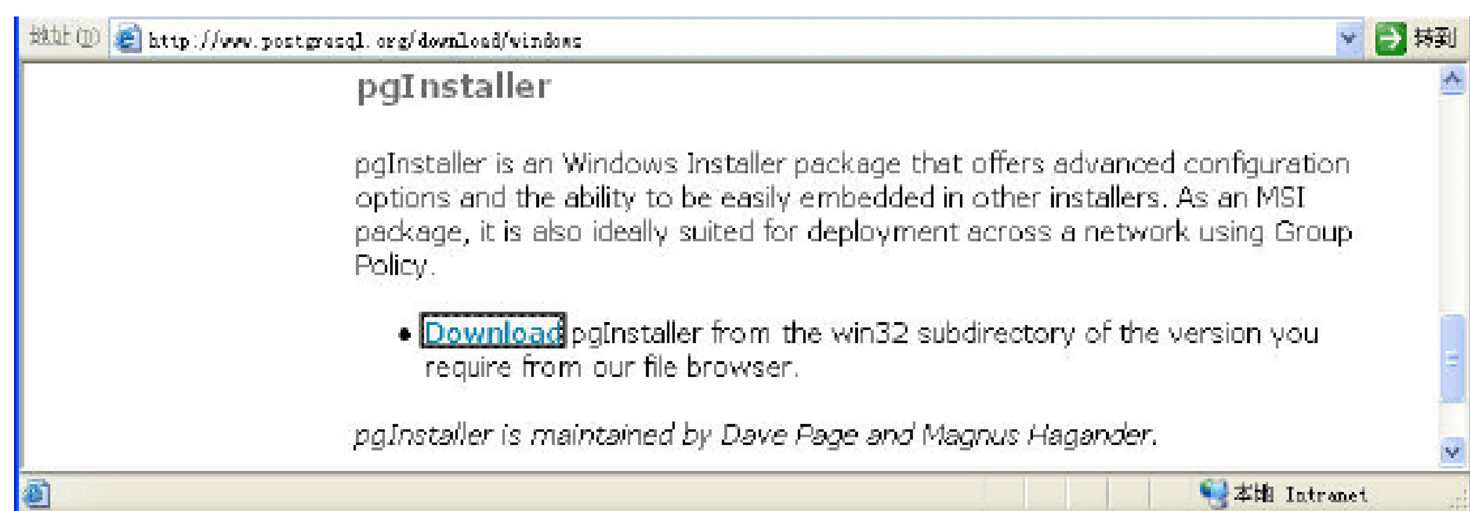


图 27.56　相关下载页面

（3）在关于 PostgreSQL 数据库下载的真正页面中（如图 27.57 所示），单击 v8.3.7 链接就可以转到选择相应安装环境的页面。

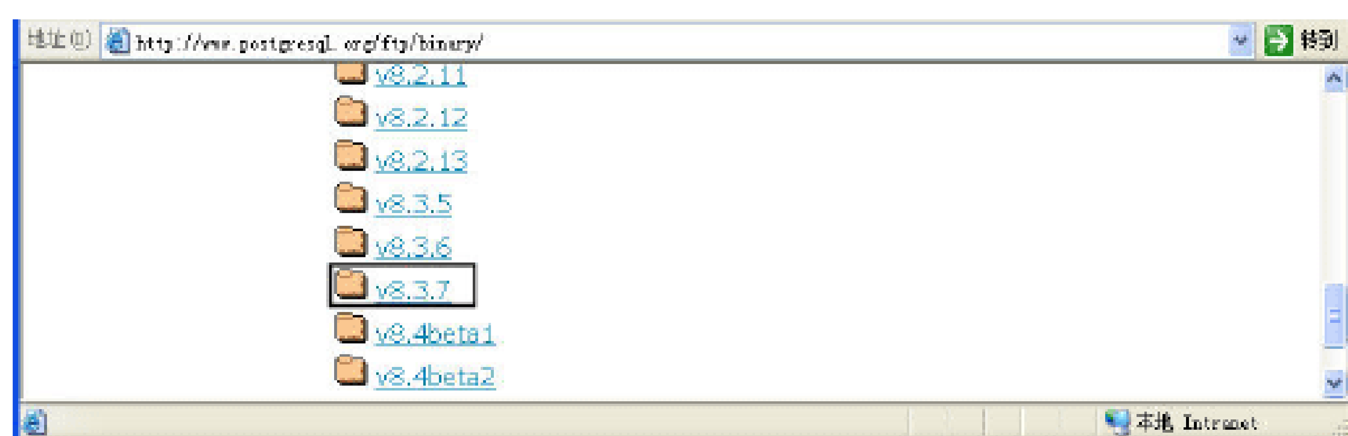


图 27.57　选择相应版本的页面

（4）在选择相应安装环境的页面中（如图 27.58 所示），单击 win32 链接就可以转到下载页面（如图 27.59 所示）。在该页面中单击 postgresql-8.3.7-1.zip 链接就可以实现该软件的下载。



图 27.58　选择安装环境

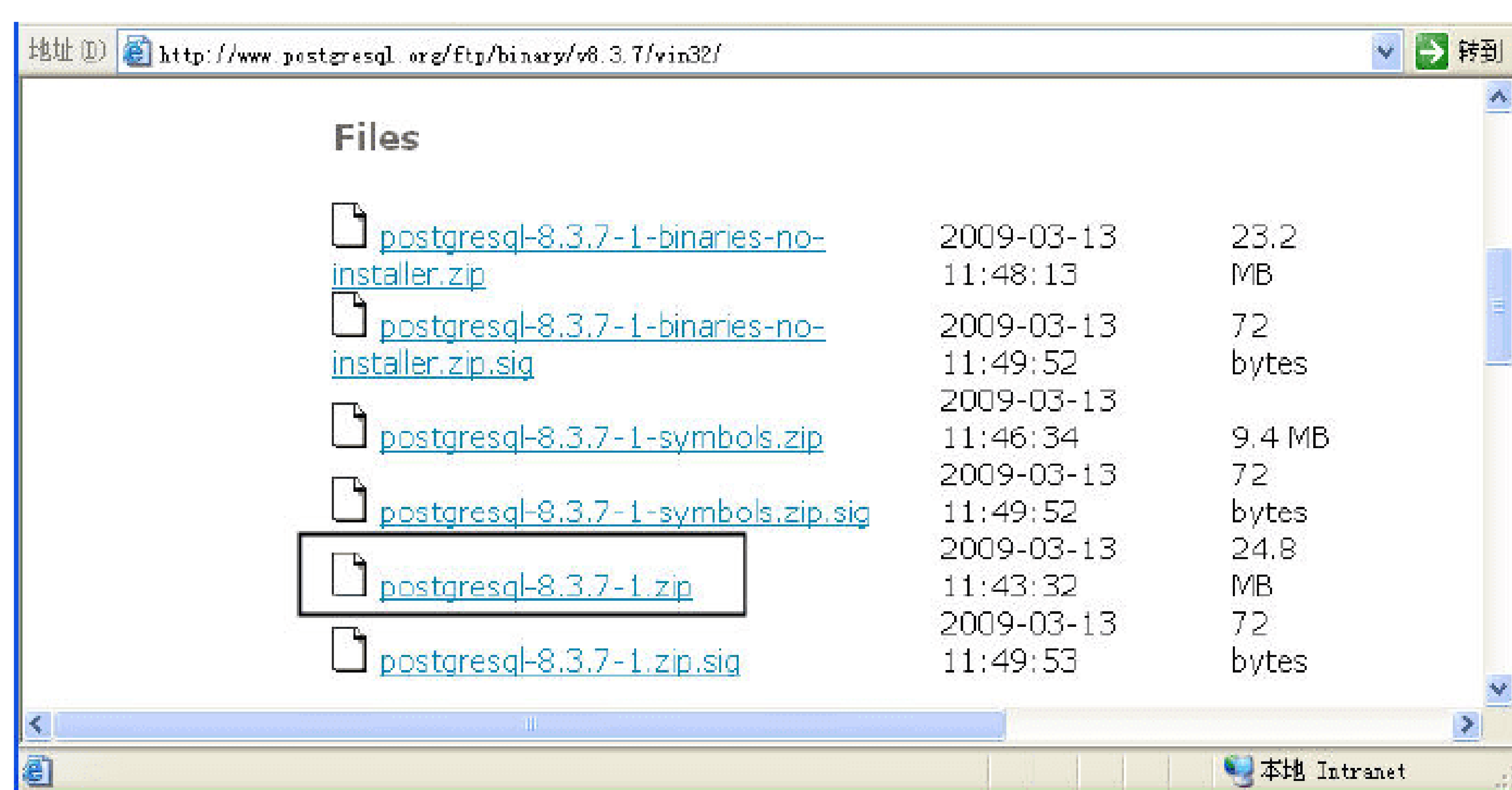


图 27.59　下载 Postgresql 数据库

如果想在 Java Web 项目中连接 PostgreSQL 数据库，则必须从 http://jdbc.postgresql.org/网站上下载与 Postgresql8.3.7 相对应的关于 JDBC 的驱动程序。

至此，就完成了对 PostgreSQL 数据库的下载。

27.6.2　安装 PostgreSQL 数据库

27.6.1 节介绍了如何下载 PostgreSQL 数据库，本节将介绍如何安装该数据库管理系统。那么如何安装 PostgreSQL 数据库管理系统呢？具体步骤如下。

（1）双击 PostgreSQL 安装程序（postgresql-8.3.7-1-windows.exe），接着就会使用 Windows Installer 开始安装过程进入欢迎界面，如图 27.60 所示。

（2）在欢迎界面中选择 Simplified Chinese 单选按钮，单击 Start 按钮弹出安装向导对话框（如图 27.61 所示）。在该对话框中单击“前进”按钮就可以进入“安装注记”对话框（如图 27.62 所示），在该对话框中会显示出 PostgreSQL 数据库的安装信息。



图 27.60　欢迎界面

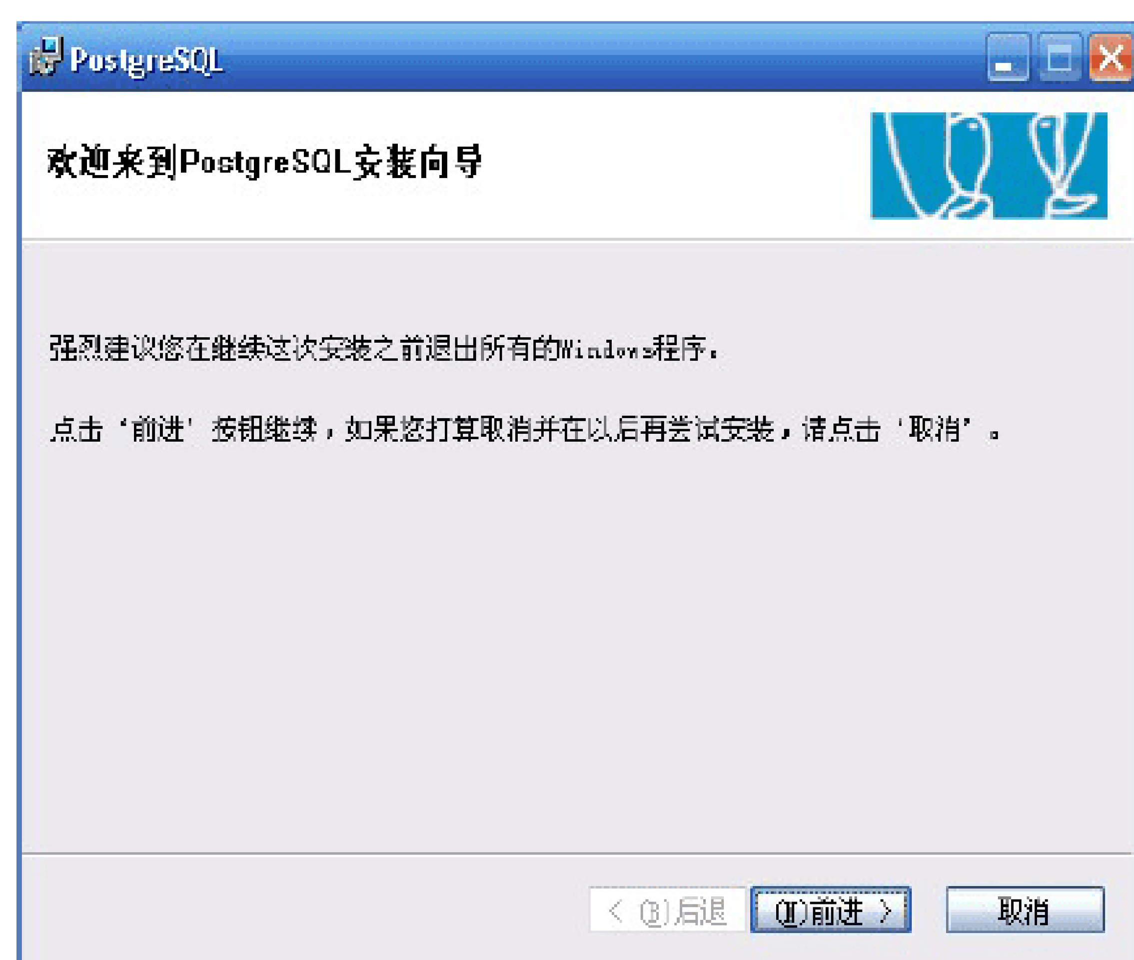


图 27.61　安装向导



图 27.62　安装注记

（3）通过单击“安装注记”对话框中的“前进”按钮，就可以进入“安装选项”对话框（如图 27.63 所示），在该对话框中可以选择安装功能和安装路径。通过单击“前进”按钮可以进入“服务配置”页面，如图 27.64 所示，在该对话框中需要填写各种选项。



图 27.63　安装选项

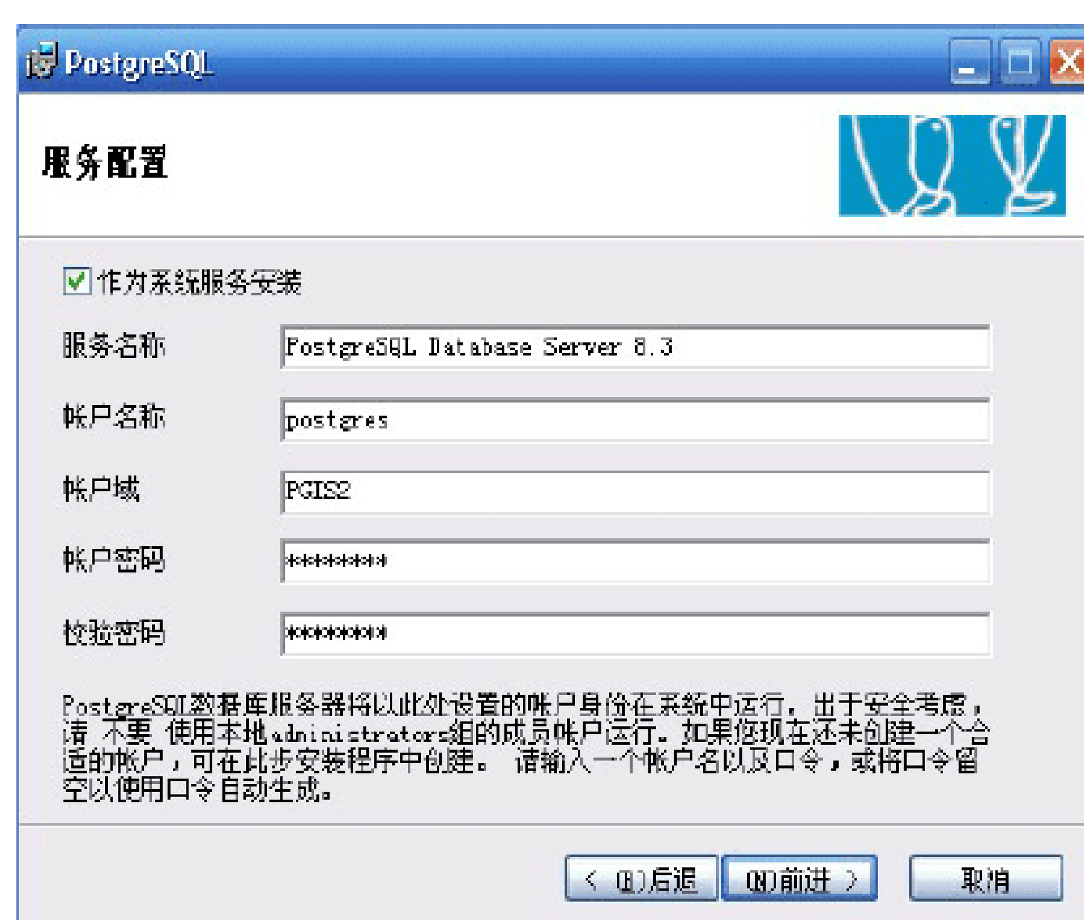


图 27.64　服务配置

🔔注意：当填写的账户密码太简单时，就会出现如图 27.65 所示的“口令”对话框。

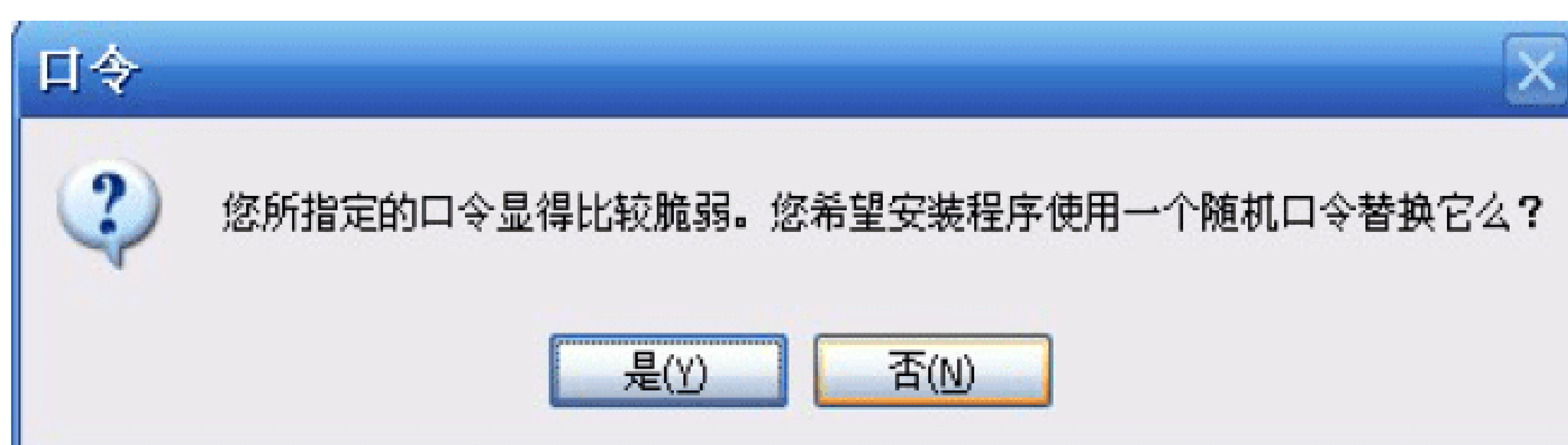


图 27.65　对话框

（4）配置完“服务配置”对话框后，单击“前进”按钮就可以进入“初始化数据库群”对话框（如图 27.66 所示），在该对话框中需要填写初始化数据库群的各项。设置完该对话框后还需要进入“设置过程语言”对话框（如图 27.67 所示）。除了设置过程语言外，还需要进入“设置辅助模块”对话框（如图 27.68 所示），设置相应的辅助模块。



图 27.66　初始化数据库群对话框



图 27.67　设置过程语言对话框

（5）设置完辅助模块后，就会进入“准备安装”界面（如图 27.69 所示）。在该界面

中单击“前进”按钮就可以进行安装了。



图 27.68　设置辅助模块对话框



图 27.69　准备安装对话框

27.7　小　　结

本章主要介绍了一个完整的商业银行设备巡检系统，该系统与市场上流行的 OA 系统极为相似，基于 Struts 2.x+Spring+Hibernate 经典框架构建而成。

在具体实现商业银行设备巡检系统时，从 Java EE 开发标准的 4 层结构体系详细讲解了该系统的各个模块：系统管理应用、设备报修管理和设备巡检管理。其中 Struts 2.0 框架实现表示层页面的跳转；Hibernate 框架实现由数据库记录转变成 POJO 对象的持久层；Spring 框架主要实现该系统业务逻辑的服务层。最后，还详细讲解了关于数据库 PostgreSQL 的相关知识。